HiFlash: Communication-Efficient Hierarchical Federated Learning With Adaptive Staleness Control and Heterogeneity-Aware Client-Edge Association

Qiong Wu, Xu Chen, *Senior Member, IEEE*, Tao Ouyang, Zhi Zhou, *Member, IEEE*, Xiaoxi Zhang, *Member, IEEE*, Shusen Yang, *Senior Member, IEEE*, and Junshan Zhang, *Fellow, IEEE*

***Abstract*—Federated learning (FL) is a promising paradigm that enables collaboratively learning a shared model across massive clients while keeping the training data locally. However, for many existing FL systems, clients need to frequently exchange model parameters of large data size with the remote cloud server directly via wide-area networks (WAN), leading to significant communication overhead and long transmission time. To mitigate the communication bottleneck, we resort to the hierarchical federated learning paradigm of HiFL, which reaps the benefits of mobile edge computing and combines synchronous client-edge model aggregation and asynchronous edge-cloud model aggregation together to greatly reduce the traffic volumes of WAN transmissions. Specifically, we first analyze the convergence bound of HiFL theoretically and identify the key controllable factors for model performance improvement. We then advocate an enhanced design of HiFlash by innovatively integrating deep reinforcement learning based adaptive staleness control and heterogeneity-aware client-edge association strategy to boost the system efficiency and mitigate the staleness effect without compromising model accuracy. Extensive experiments corroborate the superior performance of HiFlash in model accuracy, communication reduction, and system efficiency.**

***Index Terms*—Client-edge association, federated learning, hierarchical mechanism, staleness control.**

I. Introduction

NOWADAYS, federated learning (FL) has gained growing attention as it collaboratively trains a global machine learning (ML) model in distributed manner without exposing the data from private clients [1], [2]. During the training procedure of FL, the (local/global) model updates are iteratively exchanged between clients and the cloud server until reaching a desirable accurate model, thus it achieves a privacy-preserving learning by leaving training data on local clients. Various popular AI applications such as computer vision [3], language processing [4] and human activity recognition [5] have been derived within this framework.

For many existing FL systems, regardless of synchronous update (e.g., FedAvg [1] and its variants [6], [7]) or asynchronous update (e.g., FedAsync [8]), massive model parameters need to be exchanged in multiple update iterations. However, clients geographically scattered over the edges of networks are usually connected to a remote cloud server through wide-area networks (WAN) and long-distance transmissions, which would incur high communication cost and serious network congestion. Such communication inefficiency would greatly deteriorate the system performance of large-scale distributed training and further hinder the wide deployment of FL systems in practice. Hence, the research issue of boosting the communication efficiency of FL has recently drawn great attention [9].

Hierarchical architecture is a promising solution to alleviate the huge communication pressure of the cloud server, since an order of magnitude fewer data-size of model update would be transferred to cloud by aggregating local models at the lower layer in advance. Due to the merits of mobile edge computing (MEC) in practice, edge nodes (e.g., 5G edge servers) can be set as the intermediates for local model aggregation [10]. The rationales are as follows: 1) due to shorter routing path and less hop distance in the local-area network (LAN), a lower network delay and reduced network jitter are offered in the edge layer [11]. Further, the straggler problem caused by less effective communication between cloud and clients can be significantly alleviated; 2) compared to high monetary cost of WAN usage in traditional FL, abundant cheaper LAN resources at the edge nodes promote FL deployment in reality [12]; 3) FL applications are commonly scattered over massive devices, which are naturally clustered into many edge domains (e.g., campus and hospital). This distributed pattern can be well accommodated in hierarchical FL.

Manuscript received 9 June 2022; revised 30 December 2022; accepted 8 January 2023. Date of publication 19 January 2023; date of current version 24 March 2023. This work was supported in part by the National Science Foundation of China under Grants U20A20159, 61972432; in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2021B151520008; in part by the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant 2017ZT07X355. Recommended for acceptance by S. Wang. *(Corresponding author: Xu Chen.)*

Qiong Wu, Xu Chen, Tao Ouyang, Zhi Zhou, and Xiaoxi Zhang are with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, Guangdong 510006, China (e-mail: wuqiong23@mail2.sysu.edu.cn; chenxu35@mail.sysu.edu.cn; ouyt9@mail2.sysu.edu.cn; zhouzhi9@mail.sysu.edu.cn; zxx1121xiaoxi@gmail.com).

Shusen Yang is with the School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China (e-mail: shusenyang@mail.xjtu.edu.cn).

Junshan Zhang is with the ECE Department, University of California, Davis, CA 95616 USA (e-mail: jazh@ucdavis.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TPDS.2023.3238049, provided by the authors.

Digital Object Identifier 10.1109/TPDS.2023.3238049

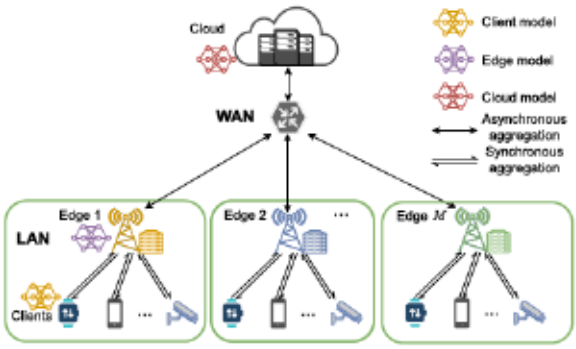


Fig. 1. Overview of HiFL approach.

Motivated by these facts, a new paradigm of client-edge-cloud hierarchical FL has recently been put forward [13], [14], which involves two levels of synchronous model aggregations, i.e., client model aggregation controlled by the edge nodes at lower layer, edge model aggregation controlled by the cloud server at higher layer. This framework aims at leveraging the advantage of synchronous update to train global model with high accuracy and fast convergence at the lower layer, benefited from high LAN bandwidth and sufficient computation resources at the edge nodes. However, a severe straggler problem at the higher layer would be incurred due to the edge heterogeneity (e.g., diverse WAN connection conditions and heterogeneous edge aggregation time due to different client size) and the communication bottleneck from edge to cloud. More explicitly, large waiting time in synchronous global model aggregation at the higher layer is inevitable.

To fully unleash the benefits of hierarchical FL, in this paper we incorporates the merits of synchronous and asynchronous operations in different aggregation layers into the hierarchical FL, which we call HiFL in order to differentiate it from HierFAVG, the version of hierarchical FL with two levels of synchronous model aggregations. As depicted in Fig. 1, confronted with huge edge heterogeneity and complicated WAN environment among edge nodes and cloud, asynchronous update is adopted for edge-cloud model aggregation to improve learning efficiency via wait-free communication. At the lower layer, synchronous model aggregation between clients and edge nodes ensures high accuracy and fast convergence. Moreover, benefited from high LAN bandwidth and sufficient computation resources at the edges in the communication-efficient one-hop access edge network environment, the straggler problem is very mild and can be neglected during synchronous client-edge aggregation, compared with the asynchronous edge-cloud aggregation communications over the latency-significant WAN.

Nevertheless, HiFL also brings in new challenges on account of the asynchronous aggregation and hierarchical mechanism design. On one hand, staleness effect arised in asynchronous update negatively impacts on the model accuracy and convergence speed [15]. Existing staleness-tolerant mechanisms usually dampen the impacts of stale model updates by only controlling the trade-off between convergence rate and variance reduction according to the staleness [8]. However, its impact on system efficiency (e.g., training time, resource efficiency) is much less considered. For example, a model with large staleness may marginally contribute to the global model, which results in more rounds of communication to reach a target accuracy for asynchronous FL. Thus, it is critical to control the model staleness for communication-efficient model learning. On the other hand, the hierarchical mechanism introduces data heterogeneity among edges, which can be further amplified by the hierarchical client-edge-cloud model aggregation and lead to degraded model performance [15]. Besides, the resource heterogeneity among the edge-associated clients can exacerbate the straggler effects, possibly prolonging the waiting time in client-edge model aggregation.

To cope with the above challenges, in this paper we first investigate HiFL to gain useful theoretical insights about its performance bound and system efficiency, and then identify the key controllable parameters that affect the learning performance. Motivated by our theoretical results, we devise an adaptive staleness control strategy for edge-cloud layer and a heterogeneity-aware association mechanism for client-edge layer to improve the overall efficiency of HiFL. For staleness control, existing approaches usually assume a pre-defined fixed threshold for the participating clients, which can not well adapt to the realistic dynamic environment. Moreover, the threshold determination is non-trivial due to the complicated FL environments (e.g., data and resource heterogeneity of clients, current running stages of FL model). Differently, we resort to the deep reinforcement learning (DRL) method and design a DRL agent based on Deep Q-Network (DQN) [16] to wisely make adaptive staleness threshold decisions tailored to the dynamic and complicated FL environments. The DRL agent is trained through a Double DQN for increased efficiency and robustness. For client-edge association, we devise an efficient weighted heuristic to find a near optimal solution that jointly minimizes the data heterogeneity among the edges and resource heterogeneity in the edge-associated clients.

In summary, this paper makes the following contributions:

- To achieve communication-efficient and accurate model learning, we resort to HiFL, a hierarchical federated learning approach that performs synchronous client-edge model aggregation and asynchronous edge-cloud model aggregation. Rigorous theoretical analysis for the convergence of HiFL is provided, including both convex and non-convex learning objectives.
- Inspired by the theoretical convergence analysis, we further advocate an enhanced design of HiFlash, which introduces adaptive staleness control and heterogeneity-aware client-edge association based on HiFL. The HiFlash approach enables large-scale deployment with boosted model performance and system efficiency.
- We devise a DRL agent based on a Deep Q-Network (DQN) for adaptive staleness control with elaborative learning reward design in order to improve system efficiency without compromising model accuracy. To mitigate the accuracy degradation and straggler effect caused by data and resource heterogeneity, we establish an efficient weighted heuristic of low-complexity for client-edge association that

well balances the trade-off between model accuracy and system efficiency.
- Extensive experiments are conducted using three widely adopted image classification datasets to evaluate the effectiveness of HiFlash, demonstrating that HiFlash significantly outperforms other FL based approaches in communication efficiency without compromising model accuracy. For example, even under highly skewed data distributions among clients, HiFlash can still achieve a high model accuracy, and meanwhile greatly reduces communication overhead, e.g., with a reduction ratio of 42% and 89% over the benchmarks of HierFAVG and FedAvg, respectively.

The rest of this paper is organized as follows: Section II presents the preliminaries on FL and DRL. Section III introduces a hierarchical FL approach named HiFL. In Section IV, we provide theoretical analysis for HiFL, and further devise HiFlash, an enhanced HiFL with adaptive staleness control and heterogeneity-aware client-edge association in Section V. Extensive experiments are conducted in Section VI. We review the related work in Section VII and conclude the paper in Section VIII.

II. Preliminaries

A. Federated Learning

Federated learning [1], first proposed by Google in 2016, trains a global shared model among massive clients in a privacy-preserving manner, where a central server serving as an aggregator coordinates client model learning. In general, a FL system consists one cloud server and N dispersed clients. Each client k has a collection of local dataset $\mathcal{D}_k = \{\mathrm{x}_j, y_j\}_{j=1}^{|\mathcal{D}_k|}$, where x_j is the feature of training sample j and y_j is its ground-truth label. To collaboratively train a global ML model $\omega \in \mathbb{R}^d$, its loss function associated with the data sample (x_j, y_j) is denoted as $f(\mathrm{x}_j, y_j, \omega)$, where d is the total number of trainable parameters. For ease of exposition, we use $f_j(\omega)$ to replace $f(\mathrm{x}_j, y_j, \omega)$ notation. As a result, the learning objective of FL is to minimize the loss function over the collection of training data at N clients, i.e.,

$$\min_{\omega \in \mathbb{R}^d} F(\omega) = \sum_{k=1}^{N} \frac{|\mathcal{D}_k|}{|\mathcal{D}|} F_k(\omega), \tag{1}$$

where $F_k(\omega) = \frac{1}{|\mathcal{D}_k|} \sum_{j \in \mathcal{D}_k} f_j(\omega)$ is the loss empirical objective over the data samples at client k, which is task-specified, for example, the learning objective can be cross-entropy loss for image classification tasks. Assuming $\mathcal{D}_k \cap \mathcal{D}_{k'} = \emptyset$ for $k \neq k'$, we define $\mathcal{D} = \cup_{k=1}^{N} \mathcal{D}_k$ and use $|\cdot|$ to denote the size of a set.

To solve the optimization problem in (1), FedAvg, the most widely used FL framework, proposes to run local stochastic gradient descent (SGD) in parallel on a sampled subset of clients and conducts synchronous model aggregation via a central server once in a while [1]. The process is repeated until the model reaches a desired accuracy. Due to slow and expensive network connection (e.g., frequent backhaul and long communication distance) between the cloud server and the geographically distributed clients [10], FedAvg performs multiple local learning steps before uploading the model updates into the cloud server, so that the number of communication rounds is considerably reduced and the network burden is further relieved.

B. Deep Reinforcement Learning

In reinforcement learning (RL), a RL agent interacts with the environment in discrete time slots to maximize its reward in the long run. At each time slot i, the RL agent observes state s^i, executes action a^i and receives a reward r^i from the environment. The state s^i of the environment then transits to s^{i+1} for the action decision making of next time slot. The whole process follows a Markov Decision Process (MDP) [17] which can be described as a tuple $\mathcal{M} =< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ wherein $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space and $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a probabilistic transition function. $\mathcal{R}$ is the immediate reward function and $\gamma \in [0, 1]$ is a factor discounting the future rewards. The objective of the RL agent is to learn a policy π^*, a mapping between states and actions that maximizes the cumulative discounted reward $R = \sum_{i=1}^{I} \gamma^{i-1} r^i$, where I is the total running slots.

To estimate the expected cumulative discounted reward starting from state s^i, value-based RL approaches adopt an action-value function

$$\begin{aligned} Q_\pi(\mathrm{s}^i, a^i) &= \mathbb{E}_\pi \left[\sum_{\hat{i}=1}^{\infty} \gamma^{\hat{i}-1} r^{i+\hat{i}-1} | \mathrm{s}^i, a^i \right] \\ &= \mathbb{E}_\pi [r^i + \gamma Q_\pi(\mathrm{s}^{i+1}, a^{i+1}) | \mathrm{s}^i, a^i], \end{aligned} \tag{2}$$

where π is the state-action mapping policy. The optimal action-value function $Q^*(\mathrm{s}^i, a^i)$ is defined as the maximum expectation of the cumulative discounted reward:

$$Q^*(\mathrm{s}^i, a^i) = \mathbb{E}_\pi [r_i + \gamma \max_a Q^*(\mathrm{s}^{i+1}, a) | \mathrm{s}^i, a^i]. \tag{3}$$

Hence, we could apply function approximation techniques to learn the action-value funtion $Q_\pi(\mathrm{s}^i, a^i, \theta_i)$ approximating the optimal function Q^*.

Nevertheless, for many real-world problems, the state space becomes too large to keep track of all the Q-values. To alleviate this issue, deep reinforcement learning (DRL) proposes to adopt DNN as the approximator of the action-value function Q_π by leveraging the powerful generalization abilities of DNNs. For example, Deep Q-Network (DQN) [16] uses a DNN to estimate the Q-values of states and actions, and the objective of DQN is minimizing the mean-squared error (MSE) loss between the target $r^i + \gamma \max_{a^{i+1}} Q_\pi(\mathrm{s}^{i+1}, a^{i+1}, \theta_i)$ and the approximator described as follows:

$$\begin{aligned} \arg\min_{\theta_i} \mathbb{L}(\theta_i) = (r^i + \gamma \max_{a^{i+1}} Q_\pi(\mathrm{s}^{i+1}, a^{i+1}, \theta_i) \\ - Q_\pi(\mathrm{s}^i, a^i, \theta_i))^2. \end{aligned} \tag{4}$$

For ease of convergence, DQN transforms DRL as a form of supervised learning and induces experience replay [18], which contains abundant transition samples, for correlation reduction between samples.

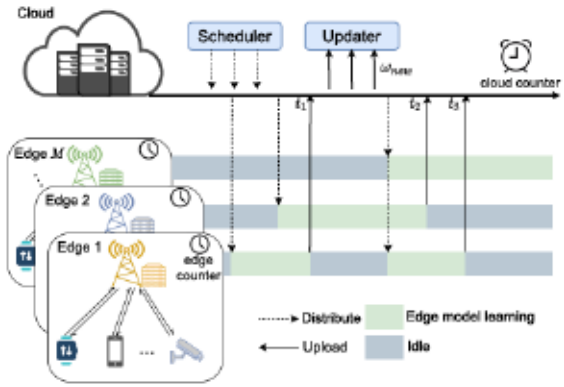


Fig. 2. Model training procedure of HiFL.

TABLE I
LIST OF KEY NOTATIONS

Symbol	Description
In General Federated Learning Settings	
N	the number of clients
M	the number of edges
$\mathcal{D}_k$	the dataset of client k
$\mathcal{C}^m$	the set of clients associated with edge m
t_e, t_c	edge counter and cloud counter
η	learning rate of federated learning
$\omega(t_c)$	the cloud model at t_c-th global iteration
$\omega^m(t_c, t_e)$	edge model at t_e-th iteration computed based on $\omega(t_c)$
$\omega_k(t_c, t_e)$	client model at t_e-th iteration computed based on $\omega(t_c)$
In Client-edge Model Aggregation	
H	the number of client-edge model aggregation
e	the number of learning epochs on the client before synchronizing with the edge node
In Edge-cloud Model Aggregation	
τ	the staleness of the edge model
α, υ	initial model weight of the edge model and penalty coefficient for calculating mixing hyperparameter α_τ in cloud model aggregation
In Adaptive Staleness Control	
$\mathbf{s}^i, a^i$	the state and action of DRL agent in time slot i
r^i	total system cost at time slot i
σ_1, σ_2	penalty factors for computation and communication costs
In Heterogeneity-aware Association	
λ	weighting parameter for data and resource heterogeneity
$L^{m,k}$	response latency for client k associated with edge m

III. HiFL Design

A. Problem Definition and HiFL Overview

We first provide the problem definition of HiFL based on the traditional FL. As depicted in Fig. 2, client model updates are not directly sent to the cloud but edge nodes. More explicitly, N participant clients are divided into M disjoint groups based on their characteristics (e.g., geographical locations), each of which is associated with one edge node. In general, the number of edge nodes is far less than clients, i.e., $M \ll N$. We denote $\mathcal{C}^m$ as the client set of edge node m, and total participant clients can be defined as $\mathcal{N} = \cup_{m=1}^{M} \mathcal{C}^m$. Thus, based on this hierarchical FL architecture, the learning objective in (1) is extended as:

$$\min_{\omega \in \mathbb{R}^d} F(\omega) = \sum_{m=1}^{M} \frac{|\mathcal{D}^m|}{|\mathcal{D}|} F^m(\omega), \tag{5}$$

where $F^m(\omega) = \sum_{k \in \mathcal{C}^m} \frac{|\mathcal{D}_k|}{|\mathcal{D}^m|} F_k(\omega)$ denotes the objective on edge node m, which is a linear combination of the empirical objectives of clients in $\mathcal{C}^m$. $|\mathcal{D}^m| = \sum_{k \in \mathcal{C}^m} |\mathcal{D}_k|$ is the data size of all samples across the clients associated with edge node m. Table I lists the key notations in our paper.

Based on the disparate behaviors of cloud and edges, HiFL adopts synchronous client-edge model aggregation and asynchronous edge-cloud model aggregation to synergistically train a high-quality ML model in a cost-efficient way. At the side of edge, high LAN bandwidth and reduced network jitter considerably shorten the communication latency for gradient exchanges or model download. Within the same LAN environment, synchronous model aggregation is more desired between edge nodes and clients, due to its high model training precision and fast convergence speed. However, at the side of cloud, model training suffers from communication bottleneck in the complicated WAN environment (e.g., highly fluctuating long-distance transmission time and diverse edge model aggregation time due to different size of clients with different edges), which leads to severe straggler problem. Thus, asynchronous aggregation is adopted to mitigate this straggler effect via reducing the waiting time of model updates between the central server and the edge nodes.

As shown in Fig. 2, we design two core components, i.e., scheduler and updater, running asynchronously in parallel on the cloud server to achieve the wait-free goal, where the former one is in charge of latest model distribution (which can be integrated with control functionality by DRL agent specified later on), and the latter one is for global model aggregation. More explicitly, once an idle edge node gets engaged in model training for its interest, it will actively inform the cloud server to download the latest version of global model. Then the cloud server will check its updater and immediately send the result to corresponding edge node. Receiving the global model, the edge node quickly broadcasts it to the associated clients and leverages their local datasets to collaboratively train a shared edge model in a synchronous manner with efficient client-edge communication. If the cloud server receives a trained model from an edge node, the updater will update the global model immediately, without waiting for other edge nodes. In order to control the model staleness caused by asynchronous aggregation, the updater conducts model aggregation with a weight penalty on the received model update, which will be elaborated in Section III-B.

Note that to improve the throughput of HiFL, multiple edge model training processes can be executed in parallel, which results in multiple updater threads with read-write lock on the global model. This asynchronous model aggregation strategy relieves the network congestion on the cloud side and enables wait-free global model learning, further reducing the communication overhead and speeding up the model training process. Particularly, different counters are set to record the model update times in this hierarchical settings, since the clients update the edge model without the coordination of the cloud server. Thus, we design cloud counter t_c and edge counter t_e for asynchronous

cloud model aggregation and synchronous edge model aggregation, respectively.

B. HiFL Training Process

The learning process of HiFL contains two main procedures, including 1) client-edge model aggregation, and 2) edge-cloud model aggregation, as elaborated below.

1) Synchronous client-edge model aggregation: When edge node m receives current global model $\omega(t_c)$ from the cloud server, the edge model is initialized as $\omega^m(t_c, t_e) = \omega(t_c)$ with $t_e = 0$, where t_c indicates the number of global aggregations conducted on the cloud model, t_e denotes the number of local model updates for edge model aggregation after receiving cloud model $\omega(t_c)$.[1] The edge model is further sent to the associated clients for client model update.

We adopt the widely used FedAvg algorithm [1] to collaboratively train a satisfactory edge model. Specifically, at t_e-th iteration, client $k \in \mathcal{C}^m$ performs model update with its local data. To reduce communication overhead, the classical FedAvg method aggregates all client models associated with edge node m and synchronizes with edge model ω^m after every c steps of local updates on each client. Denote $\omega_k^m(t_c, t_e)$ as the model parameters of client $k \in \mathcal{C}^m$, then $\omega_k^m(t_c, t_e)$ evolves in the following way:

$$\omega_k^m(t_c, t_e) = \begin{cases} \omega_k^m(t_c, t_e-1) - \eta \nabla F_k(\omega_k^m(t_c, t_e-1)), & t_e \bmod c \neq 0 \\ \omega^m(t_c, t_e), & t_e \bmod c = 0 \end{cases} \tag{6}$$

where

$$w^m(t_c, t_e) = \sum_{k \in \mathcal{C}^m} \frac{|\mathcal{D}_k|[\omega_k^m(t_c, t_e-1) - \eta \nabla F_k(\omega_k^m(t_c, t_e-1))]}{|\mathcal{D}^m|}. \tag{7}$$

Without loss of generality, we assume that the edge node performs a number of H model aggregations (e.g., $t_e \leq Hc$) in each global training round, which indicates that Hc client model updates have been performed for one client during the client-edge model aggregation. After this collaborative model training, the edge model is updated as $w^m(t_c, Hc)$ and will asynchronously update the global model with the cloud server.

2) Asynchronous edge-cloud model aggregation: The asynchronous mechanism in edge-cloud model aggregation introduces the challenge of staleness as multiple edge nodes are free to perform model training and uploading at arbitrary times. For example, at the global counter t_c, the cloud server receives a stale model $\omega^m(t_c - \tau, Hc)$ which is trained by edge node m based on the global model $\omega(t_c - \tau)$, where τ represents the staleness of the edge model. As the edge model is trained based on an outdated cloud model version, the stale model will add noise to the cloud model training procedure, slow down or even prevent the training convergence [19].

[1]The maximum value of t_c can be determined by the cloud server based on the convergence of the model, while the maximum value of t_e can be set by each edge node and hence is different across various nodes.

To control the error caused by asynchrony, HiFL updates the global model with the stale edge model by introducing a mixing hyperparameter α_τ as in [8],

$$\omega(t_c) = (1 - \alpha_\tau)\omega(t_c - 1) + \alpha_\tau \omega^m(t_c - \tau, Hc), \tag{8}$$

where α_τ is the weight that the edge model $\omega^m(t_c - \tau, Hc)$ with staleness τ contributes to the global model. A smaller α_τ will result in more FL training rounds while a bigger value of α_τ can cause large accuracy fluctuation. By adjusting the value of α_τ, we can adaptively control the trade-off between convergence speed and variance reduction in the model learning process. In this paper, we use the following exponential function to determine the value of α_τ,

$$\alpha_\tau = \alpha \cdot \upsilon^\tau, \tag{9}$$

where $\alpha \in (0, 1)$ is the initial model weight of the edge model. We can decrease α to mitigate the error caused by large staleness τ with the penalty coefficient $\upsilon \in (0, 1)$.

The cloud server and edge nodes in HiFL conducts model updates asynchronously until the cloud model converges. The synchronous client-edge aggregations on different client groups can be conducted in parallel, and the asynchronous edge-cloud model aggregation avoids from long waiting time, both of which contribute to the fast model learning and wait-free communication. The details of the HiFL algorithm is elaborated in Algorithm 1.

IV. Convergence Analysis

A. Definitions and Assumptions

For the purpose of the analysis, we introduce the following definitions and assumptions to the loss function.

Assumption 1: (Smoothness). The function $F_k(\omega)$ is β-smooth if $\forall \omega, \omega'$,

$$||\nabla F_k(\omega) - \nabla F_k(\omega')|| \leq \beta ||\omega - \omega'||, \tag{10}$$

where $\beta > 0$.

Assumption 2: (Strong convexity). The function $F_k(\omega)$ is μ-strongly convex if $\forall \omega, \omega'$,

$$\langle \nabla F_k(\omega'), \omega - \omega' \rangle + \frac{\mu}{2}||\omega - \omega'||^2 \leq F_k(\omega) - F_k(\omega'), \tag{11}$$

where $\mu \geq 0$. Note that if $\mu = 0$, $F_k(\omega)$ is convex.

Assumption 3: (Weak convexity). The function $F_k(\omega)$ is μ-weakly convex if the function $G_k(\omega) = F_k(\omega) + \frac{\mu}{2}||\omega||^2$ is convex, where $\mu \geq 0$. Specifically, $F_k(\omega)$ is convex if $\mu = 0$ and potentially non-convex if $\mu > 0$.

Assumption 4: (Lipschitz). The function $F_k(\omega)$ is ρ-Lipschitz if $\forall \omega, \omega'$,

$$||F_k(\omega) - F_k(\omega')|| \leq \rho ||\omega - \omega'||. \tag{12}$$

Under these assumptions, Lemma 1 holds for the loss functions of the edge models and the cloud model.

Lemma 1: $F^m(\omega)$ and $F(\omega)$ are μ-strongly convex, β-smooth and ρ-Lipschitz.

Proof: It is straightforward from the aforementioned assumptions, the definition of $F^m(\omega)$, $F(\omega)$ and triangle inequality.

Algorithm 1: HiFL Training Procedure.

Input: Datasets from N distributed clients $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$, number of local updates c, and learning rate η.

1: Conduct client clustering based on some predefined criteria (e.g., geographical locations) or by heterogeneity-aware client-edge association strategy in Algorithm 3
2: **Cloud server executes:**
3: Initialize the cloud model ω, $t_c \leftarrow 0$
4: **Scheduler:**
5: Periodically distribute global model $\omega(t_c)$ and global counter t_c to one edge node for edge model update
6: **Updater:**
7: **for** $t_c = 1, 2, \ldots, T_c$ **do**
8: Receive a pair $(\omega^m(\tilde{t}_c, Hc), \tilde{t}_c)$ from one edge node
9: Calculate the model staleness $\tau = t_c - \tilde{t}_c$
10: Update cloud model with (8)
11: **end for**
12: **Edge node executes:**
13: **for** $m \in 1, 2, \ldots, M$ in parallel **do**
14: **if** received a pair of the cloud model and the cloud counter $(\omega(t_c), t_c)$ from Scheduler **then**
15: Set $\tilde{t}_c = t_c$ and $\omega^m(\tilde{t}_c, 0) = \omega(t_c)$
16: Initialize $\omega_k^m(\tilde{t}_c, 0) = \omega^m(\tilde{t}_c, 0)$ for $k \in \mathcal{C}^m$
17: **for** $t_e = 1, \ldots, Hc$ **do**
18: /*Client executes*/
19: **for** $k \in \mathcal{C}^m$ **do**
20: Calculate current client model by (6)
21: **end for**
22: **end for**
23: Send $(\omega^m(\tilde{t}_c, Hc), \tilde{t}_c)$ to the cloud server.
24: **end if**
25: **end for**

Output: Cloud model $\omega(T_c)$.

In the cloud model training process of HiFL, there are two levels of model aggregation, client-edge model aggregation and edge-cloud model aggregation, conducted in parallel. Following [6], we introduce the notion of *virtual cluster model learning* in Definition 1 to find the loss divergence between the edge model trained by synchronous client-edge model aggregation and a virtual cluster model where the training data is assumed to exist on a virtual central repository. Next, we formalize *cluster-based gradient divergence* in Assumption 5 to characterize the impact of the difference in data distributions across clients and edge nodes on HiFL.

Definition 1: (Virtual cluster model learning). For client-edge model aggregation, we use the shorthand notation $[h] = [(h-1)c, hc)$ to indicate an interval between two successive edge model aggregation. Given a certain client cluster $\mathcal{C}^m$ associated with edge node m and the initialized edge model $\omega^m(t_c, 0) = \omega(t_c)$, for any interval $[h]$, $h = 1, 2, \ldots, H$, the virtual cluster model $v_{[h]}^m(t_c, t_e)$ are updated by performing gradient descent on the centralized data examples $\mathcal{D}^m$ owned by $\mathcal{C}^m$, and synchronizes with the federated edge model ω^m at the beginning of each interval, as shown in (13),

$$v_{[h]}^m(t_c, t_e) = \begin{cases} v_{[h]}^m(t_c, t_e-1) - \eta \nabla F^m(v_{[h]}^m(t_c, t_e-1)), \\ \qquad\qquad\qquad\qquad t_e \bmod c \neq 0 \\ \omega^m(t_c, t_e), \qquad\qquad t_e \bmod c = 0. \end{cases} \tag{13}$$

Assumption 5: (Cluster-Based Gradient Divergence). For any client $k \in \mathcal{C}^m$, δ_k^m is assumed as an upper bound of the gradient difference between the local loss function of client k and the edge loss function of edge node m, which can be expresses as follows,

$$||\nabla F_k(\omega) - \nabla F^m(\omega)|| \leq \delta_k^m, \tag{14}$$

Then, we have $\delta^m \triangleq \frac{\sum_{k \in \mathcal{C}^m} |D_k| \delta_k^m}{\sum_{k \in \mathcal{C}^m} |D_k|}$ for the client cluster associated with edge node m and δ_{max} as the biggest gradient difference across $\{\delta^m\}_{m=1}^M$.

We assume Δ as an upper bound of the gradient difference between the loss function of any edge node m and that of the global loss function, i.e.,

$$||\nabla F^m(\omega) - \nabla F(\omega)||^2 \leq \Delta. \tag{15}$$

We call δ_{max} as the client-edge divergence and Δ as the edge-cloud divergence. In addition, the expected squared norm of stochastic gradients on any client k is defined to be uniformly bounded, i.e.,

$$\mathbb{E}||\nabla F_k(\omega)||^2 \leq V. \tag{16}$$

For μ-weakly convex loss function F_k (which can be non-convex if $\mu > 0$), we define $G_{\tilde{\omega}} = F(\omega) + \frac{\tilde{\mu}}{2}||\omega - \tilde{\omega}||^2$ with $\tilde{\mu} > \mu$. Similarly with the convex settings, we assume $G_{\tilde{\omega}}$ is $\tilde{\beta}$-smooth and $\tilde{\rho}$-Lipschitz. $\forall \tilde{\omega} \in \mathbb{R}_d$, we have $||\nabla G_{k,\tilde{\omega}}(\omega) - \nabla G_{\tilde{\omega}}^m(\omega)|| \leq \tilde{\delta}_k^m$, $\tilde{\delta}^m \triangleq \frac{\sum_{k \in \mathcal{C}^m} |D_k| \tilde{\delta}_k^m}{\sum_{k \in \mathcal{C}^m} |D_k|}$ for the client cluster associated with edge node m and $\tilde{\delta}_{max}$ as the biggest gradient difference across $\{\tilde{\delta}^m\}_{m=1}^M$. Furthermore, we assume $||\nabla G_{\tilde{\omega}}^m(\omega) - \nabla G_{\tilde{\omega}}(\omega)||^2 \leq \tilde{\Delta}$ and $||\nabla G_{k,\tilde{\omega}}(\omega)||^2 \leq \tilde{V}$.

B. Convergence of HiFL

Based on the assumptions and definitions above, we have the following convergence guarantees.

Lemma 2: During client-edge model aggregation, for any interval $[h]$ and $t_e \in [h]$, we have

$$||\omega^m(t_c, t_e) - v_{[h]}^m(t_c, t_e)|| \leq g(t_e - (h-1)c), \tag{17}$$

where

$$g(x) = \frac{\delta_{max}}{\beta}((\eta\beta + 1)^x - 1) - \eta\delta_{max}x, \tag{18}$$

for any $x = 0, 1, 2, \cdots$.

Furthermore, as $F^m(\cdot)$ is ρ-Lipschitz, we have $F^m(\omega^m(t_c, t_e)) - F^m(v_{[h]}^m(t_c, t_e)) \leq \rho g(t_e - (h-1)c)$.

Proof: Please refer to Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2023.3238049, of the separate supplementary file for details.

Thus, when the client-edge model aggregation finishes, e.g., $t_e = Hc$, the loss divergence between the edge model trained by FL and the virtual cluster model is $F^m(\omega^m(t_c, Hc)) - F^m(v^m_{[h]}(t_c, Hc)) \leq \rho g(c)$. With the help of the weight deviation upper bound, we are now ready to prove the convergence of HiFL for both convex and non-convex loss functions.

Theorem 1: Suppose the loss function F_k is μ-strongly convex, and each edge node executes $H \in [H_{min}, H_{max}]$ client-edge aggregations before pushing the edge model to the cloud server. Taking $\eta < \frac{1}{\beta}$, the convergence upper bound of HiFL after T_c global updates on the cloud server can be expressed as,

$$\mathbb{E}[F(\omega(T_c) - F(\omega^*)] \leq \kappa[F(\omega(0)) - F(\omega^*)] + (1-\kappa)\frac{A_1 V + A_2\delta_{max} + A_3\Delta}{B}, \tag{19}$$

where $\kappa = \left(1 - \alpha_\tau + \alpha_\tau(1-\eta\mu)^{cH_{min}}\right)^{T_c}$, $A_1 = \frac{1}{2\mu}$, $A_2 = \rho H_{max}\left(\frac{(\eta\beta+1)^c - 1}{\beta} - \eta c\right)$, $A_3 = \frac{cH_{max}\eta}{2}$ and $B = 1 - (1 - \eta\mu)^{cH_{min}}$.

Proof: Please refer to Appendix B, available in the online supplemental material, of the separate supplementary file for details.

Theorem 2: Suppose the loss function F_k is μ-weakly convex (which can be non-convex if $\mu > 0$), and each edge node executes $H \in [H_{min}, H_{max}]$ client-edge aggregations before pushing the edge model to the cloud server. Taking $\eta < \min(\frac{1}{\beta}, \frac{2}{\bar{\mu}-\mu})$, the convergence upper bound of HiFL after T_c global updates on the cloud server can be expressed as,

$$\mathbb{E}[F(\omega(T_c) - F(\omega^*)] \leq \tilde{\kappa}[F(\omega(0)) - F(\omega^*)] + (1-\tilde{\kappa})\frac{\tilde{A}_1\tilde{V} + \tilde{A}_2\tilde{\delta}_{max} + \tilde{A}_3\tilde{\Delta}}{\tilde{B}}, \tag{20}$$

where $\tilde{\kappa} = \left(1 - \alpha_\tau + \alpha_\tau\left[\left(1 - \frac{\eta(\bar{\mu}-\mu)}{2}\right)^{cH_{min}}\right]\right)^{T_c}$, $\tilde{A}_1 = \frac{12\rho^2 H_{max}}{(\bar{\mu}-\mu)^3} + \frac{5\bar{\mu}-\mu}{2(\bar{\mu}-\mu)^2}$, $\tilde{A}_2 = \frac{\bar{\rho}H_{max}}{\bar{\beta}}((\eta\bar{\beta}+1)^c - 1 - \bar{\beta}\eta c)$, $\tilde{A}_3 = \frac{2H_{max}}{\bar{\mu}-\mu}$ and $\tilde{B} = 1 - \left(1 - \frac{\eta(\bar{\mu}-\mu)}{2}\right)^{cH_{min}}$.

Proof: We first give the convergence guarantee between the client model and the virtual cluster model and then provide the details of convergence analysis for the cloud model in Appendix C, available in the online supplemental material, of the separate supplementary file.

Based on Theorems 1 and 2, we draw the following three notable remarks for the convergence of HiFL.

Remark 1: (Convergence rate.) The hyperparameter α_τ controls the convergence rate of HiFL. Since α_τ increases with the decrease of τ, if a smaller τ is adopted, κ will decrease to 0 faster as the total number of global aggregations T_c grows, indicating a faster convergence rate.

Remark 2: (Convergence bound.) When $T_c \to \infty, \kappa \to 0$, the convergence bound is reduced to $\frac{A_1 V + A_2\delta_{max} + A_3\Delta}{B}$ for strongly convex function, which is dominantly affected by the stochastic gradient of client V, the client-edge divergence δ_{max}, and the edge-cloud divergence Δ. Here, the values of the two coupled items, δ_{max} and Δ, are determined by the client-edge association strategy. Similar observations can be found for weakly convex function.

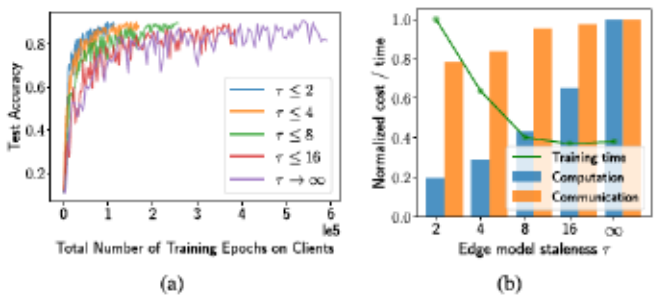


Fig. 3. The model performance of HiFL in different aspects with varying values of edge model staleness threshold τ on MNIST dataset. (a) shows that a bigger value of τ will cause huge accuracy fluctuation. (b) gives the computation cost of clients, communication cost of edges and the training time of HiFL with varying τ.

Remark 3: (Impact of τ on convergence bound.) The right side of (19) can be reformulated as $U = (C_1)^{T_c}(C_2 - C_3) + C_3$, where $C_1 = 1 - \alpha_\tau + \alpha_\tau(1-\eta\mu)^{cH_{min}}$, $C_2 = F(\omega(0)) - F(\omega^*)$ and $C_3 = \frac{A_1 V + A_2\delta_{max} + A_3\Delta}{B}$. Since C_2 is usually very large, we assume $C_2 - C_3 > 0$. Hence, U monotonically increases with C_1. As C_1 increases with τ, given a fixed value of T_c in practice, a bigger value of τ indicates a bigger upper bound U. Similar observations can be found for weakly convex function.

V. HiFlash: HiFL With Adaptive Staleness Control and Heterogeneity-Aware Client-Edge Association

In this section, with the theorectical analysis above, we first conduct a preliminary evaluation on the performance of HiFL with different model staleness values and client-edge association mechanisms. Inspired by the empirical insights from experimental results, we then devise an enhanced design of HiFL, named HiFlash, with adaptive staleness control and heterogeneity-aware client-edge association to achieve high efficiency.

A. Performance of HiFL in Deployment

We first use MNIST dataset [20] as an example to study the model training performance of HiFL under varying model staleness values from both model performance and system cost perspectives. As depicted in Fig. 3(a), $\tau \leq 8$ means that all the edge nodes have the same maximum staleness threshold, which is 8. Hence, each edge node can upload its trained edge model with different τ to the cloud if τ is less than 8. For HiFL without staleness control (e.g., $\tau \to \infty$), all the edge models uploaded by the edge nodes can be utilized for the cloud model updating, no matter the edge model staleness. As we can see, it requires 5.3×10^5 training epochs on clients to reach a target test accuracy of 0.9 for HiFL with $\tau \to \infty$, since large model staleness results in slow convergence and dramatic accuracy fluctuation. While HiFL with staleness-restricted adjustment only allows

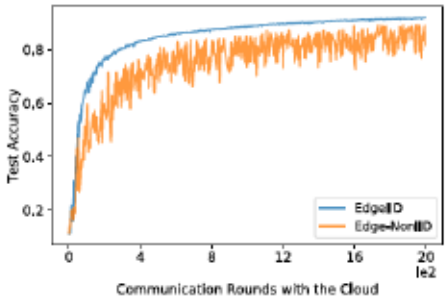


Fig. 4. EdgeIID association strategy results in smaller edge-cloud model divergence and ultimately achieves fast convergence and high model accuracy than Edge-NonIID.

edge model updates within a smaller staleness, ensuring a satisfactory convergence speed. For example, when $\tau \leq 2$, the cloud model can reach a test accuracy of 0.9 within 9.8×10^4 training epochs on clients, less than $1/5$ of the computation cost in the case of $\tau \to \infty$.

Besides, the overall system cost should be better quantified and jointly considered in realistic large-scale FL system. Thus, we introduce three performance metrics for HiFL: *total training time of the cloud*, *communication cost of the edges*, and *computation cost of the clients*. As observed in Fig. 3(b), higher communication and computation costs are incurred in HiFL without staleness control in the long run. Nevertheless, smaller τ indicates decreased model parallelism, where fewer edge nodes are allowed to simultaneously train the model, considerably prolonging the training time of FL model learning. For example, in HiFL with $\tau = 0$, an edge can perform model training only when all the others are idle. Hence, staleness control for HiFL is critical for fast and cost-efficient model learning, which should be well designed to achieve a better trade-off between training time and cost efficiency.

Since the data heterogeneity [21] can be a critical issue in FL, we study the influence of varying client-edge divergences and edge-cloud divergences on HiFL via multiple different client-edge association strategies. A useful insight is derived from the results, that is, edge-cloud divergence Δ, as the dominant factor, negatively impacts the cloud model accuracy. As shown in Fig. 4, we consider a FL system with a cloud server, 10 edge nodes and 100 clients. Each client owns samples from only one single class in MNIST dataset. Edge-IID means that the clients are clustered into different edge groups and the data distributions on the edge nodes are IID (e.g., identical number of samples from 10 classes). While in Edge-NonIID case, the samples maintained by an edge node are from 5 classes. Edge-IID association strategy groups the clients with a smaller edge-cloud model divergence, and ultimately leads to fast convergence and high accuracy. Therefore, given the clients with Non-IID distributions, a heterogeneity-aware client-edge association strategy is desired to make the data distributions on the edge nodes similar to the global IID distribution.

Motivated by the observations above, we devise HiFlash, an enhanced HiFL approach equipped with adaptive staleness control at the edge-cloud layer and heterogeneity-aware association at the client-edge layer, as elaborated below.

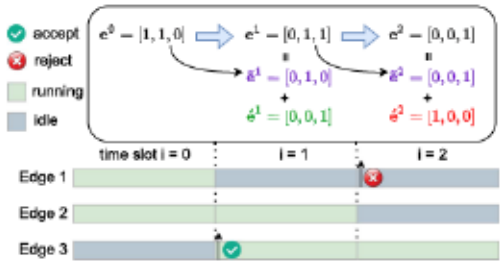


Fig. 5. The illustration of the relations among $\mathbf{e}^i$, $\tilde{\mathbf{e}}^i$ and $\acute{\mathbf{e}}^i$.

B. Adaptive Staleness Control At Edge-Cloud Layer

Fixed staleness control (e.g., $\tau <= 8$ in Fig. 3) requires a predefined staleness threshold for all the edge nodes, which can work poorly in the complex dynamic FL environment (e.g., highly dynamic communication capabilities of edges, time-varying number of current training edges) and further degrades model performance. Thus, we design an adaptive edge staleness threshold for the edge nodes which are willing to join the global model training based on the condition of its control domain (e.g., the computation resources of clients, the communication capabilities and training time of edges). Specifically, we formulate the system cost model and then adopt a deep reinforcement learning approach to dynamically control the staleness threshold.

System Cost Model: To fully characterize the environment dynamics, we adopt a slotted structure for staleness control to divide a long-term time horizon into a series of discrete time slot. Note that the length of each time slot is usually short, thus we assume there are at most one edge node sends a check-in request to the cloud at the beginning of time slot i. Similarly, at most one edge node will finish the edge model training and upload the model updates to the cloud server at the end of a time slot. We define the running/idle modes of edges at time slot i as $\mathbf{e}^i$, which is composed of the edges $\tilde{\mathbf{e}}^i$ that do not finish the edge model training task at previous time slot, and the edge $\acute{e}^i$ which sends a check-in request and is accepted by the cloud server for participating the FL training. An example of the relationship among the definitions of $\mathbf{e}^i$, $\tilde{\mathbf{e}}^i$ and $\acute{\mathbf{e}}^i$ is illustrated in Fig. 5.

1) Computation cost: At a given time slot i, we define the computation cost of an edge node m as the sum of computation cost of its associated clients:

$$C_{comp}^{i,m} = \sum_{k \in \mathcal{C}_m} C_{comp}^{i,k}, \tag{21}$$

where $C_{comp}^{i,k}$ denotes the computation cost of client k. Similarly to many existing works [22], [23], [24], following the empirical measurement study [13], we assume $C_{comp}^{i,k} = \frac{cD_k\zeta_k}{f_{i,k}}$, where $f_{i,k}$ is the processing speed of client k at time slot i, and ζ_k is processing density for client k.[2] D_k is the total number of bits for the training data of client k in one local iteration and c is the number of local iterations. Hence, the product of c and D_k

[2]It is possible to train local model with GPU for devices with GPU resources, and accordingly, the computation cost is calculated with GPU cycle frequency and GPU processing density of the devices, which can be obtained by measurements.

indicates the workload for client k. The computation cost of all clients at time slot i is denoted as

$$C^i_{comp} = \sum_{m=1}^{M} e^i_m \cdot C^{i,m}_{comp}, \tag{22}$$

where $e^i_m \in \{0, 1\}$ indicates the idle/running modes of edge node m at time slot i.

2) Communication cost: The communication cost at time slot i is denoted as

$$C^i_{comm} = \sum_{m=1}^{M} (e^i_m - \bar{e}^{i+1}_m) \cdot C^{i,m}_{comm}, \tag{23}$$

where $C^{i,m}_{comm} = \sum_{k \in \mathcal{C}_m} C^{i,m,k}_{comm}$ is the communication cost of edge node m at time slot i. Following [25], the communication cost between edge m and client k at time slot i is calculated by $C^{i,m,k}_{comm} = \frac{\mathbb{M}_\omega \times 32 \text{ bits/parameter}}{B_{i,m,k} \log_2(1+\text{SNR})}$, where $B_{i,m,k}$ is the allocated bandwidth for client k by edge node m at time slot i, $\mathbb{M}_\omega$ is the number of parameters of ω and SNR is set to be 17 dB.

The DRL Agent for Adaptive Staleness Control: The primary objective of model staleness control is to minimize the total system cost (including total training time of the cloud, communication cost of the edges, and computation cost of the clients) of HiFL system while achieving a target model training performance (e.g., a target accuracy Ω). Due to the complicated FL learning environment, we design an experience-driven algorithm based on DRL for adaptive staleness control. We first formulate the adaptive staleness threshold optimization problem as a MDP as follows:

1) State: At each time slot t, the system state is composed of three kinds of information to characterize current HiFL training environment, as elaborated below:

- The information of edge training performance consists the estimated computation cost $\tilde{\mathbf{C}}^i_{comp} = [\tilde{C}^{i,1}_{comp}, \ldots, \tilde{C}^{i,M}_{comp}]$, the estimated communication cost $\tilde{\mathbf{C}}^i_{comm} = [\tilde{C}^{i,1}_{comm}, \ldots, \tilde{C}^{i,M}_{comm}]$ and the estimated time slots $\tilde{\mathbf{T}}^i_{train} = [\tilde{T}^{i,1}_{train}, \ldots, \tilde{T}^{i,M}_{train}]$ required for each edge node to complete edge model calculation.
- The information of current running edges is characterized as the remaining training time of current edges $\tilde{\mathbf{e}}^i$, denoted as $\tilde{\mathbf{T}}^i_{rem} = [\tilde{T}^{i,1}_{rem}, \ldots, \tilde{T}^{i,M}_{rem}]$, where $\tilde{T}^{i,m}_{rem} = 0$ if edge node m is idle (e.g., $\tilde{e}^i_m = 0$).
- The information of current check-in request of the edges $\acute{\mathbf{e}}^i$ indicates the edge which will be informed of a staleness threshold by the DRL agent. Note that at most one check-in request from the idle edges happens at one time slot (e.g., $|\acute{\mathbf{e}}^i| \in \{0, 1\}$). Moreover, the edge node m that requests for check-in does not belong to the set of current running edges, which means $\sum_{m=1}^{M} \acute{e}^i_m \cdot \tilde{T}^{i,m}_{rem} = 0$.

In summary, the state can be represented as $\mathbf{s}^i = [\tilde{\mathbf{C}}^i_{comp}, \tilde{\mathbf{C}}^i_{comm}, \tilde{\mathbf{T}}^i_{train}, \tilde{\mathbf{T}}^i_{rem}, \acute{\mathbf{e}}^i]$. It is worthnoting that the estimated cost and training time information in the state can be profiled and collected jointly by the edge nodes and cloud server, such that the cloud server will be aware of the cost information of an edge node with a check-in request.

2) Action: At the beginning of each time slot i, the DRL agent needs to decide the maximum staleness a^i that can tolerate based on current state $\mathbf{s}^i$ for the idle edge node who requests for check-in ($\acute{e}^i_m = 1$). In this paper, we set an upper bound τ_{max} for staleness threshold a^i and a lower bound -1 which means the check-in request is rejected by the cloud server. Here, the rejection operation can adjust the number of running edges (control the model parallelism) and hence mitigate the straggler effect. As a consequence, the decision space of action a^i is $\{-1, 0, 1, \ldots, \tau_{max}\}$. Once an action a^i is performed, the idle/running modes of edges will be changed based on the following equation

$$\mathbf{e}^i = \begin{cases} \bar{\mathbf{e}}^i + \acute{\mathbf{e}}^i, & a^i >= 0, \\ \tilde{\mathbf{e}}^i, & a^i < 0. \end{cases} \tag{24}$$

3) Reward: We define the reward function r^i as the total system cost at time slot i:

$$r^i = -\sigma_1 C^i_{comp} - \sigma_2 C^i_{comm} - 1, \tag{25}$$

where σ_1 and σ_2 are two penalty factors for computation cost and communication cost, respectively.

The DRL agent aims to find the action (staleness threshold) which can minimize the long-term system cost, while guaranteeing a certain level of learning quality (e.g., reaching a target accuracy Ω). Thus, the problem can be formulated to maximize the expectation of the cumulative discounted reward starting from time slot i given by:

$$R^i = \sum_{\tilde{i}=1}^{I} \gamma^{\tilde{i}-1} r^{i+\tilde{i}-1} = \sum_{\tilde{i}=1}^{I} \gamma^{\tilde{i}-1} (-\sigma_1 C^{i+\tilde{i}-1}_{comp} - \sigma_2 C^{i+\tilde{i}-1}_{comm} - 1), \tag{26}$$

where $\gamma \in (0, 1]$ is a factor discounts future rewards and I is the total training slots for reaching the target model accuracy.

We now explain the motivations of the reward design. The reward r^i is defined as a weighted sum of computation cost, communication cost and training time. The first two terms incentivize the agent selects action that results in smaller communication and computation costs. The last term, -1, encourages the agent to complete training in fewer time slots (e.g., a smaller value of the total training slots I). By adjusting the non-negative parameters σ_1 and σ_2, it is able to meet diverse preferences of different FL learning tasks on resource efficiency and learning time. For example, we can assign higher weights to resource cost so that smaller staleness threshold are prefer to be chosen in fear of severe staleness effect and resource waste. While smaller weights of resource cost indicates that training time is critical for a FL training task and more decisions with bigger staleness threshold are made to facilitate more parallel model training.

Training Procedure of DRL Agent: Considering the continuous and high-dimensional space and the limited available traces from FL tasks, we adopt DQN as the DRL agent to efficiently learn the optimal staleness control policy. Due to the complicated dynamics in FL system, overestimation issue can easily arise due to insufficient exploration by DQN. Hence, to solve the overestimation problem, we propose to use Double Deep Q-learning (DDQN) to learn the approximator $Q_\pi(\mathbf{s}^i, a^i)$ that

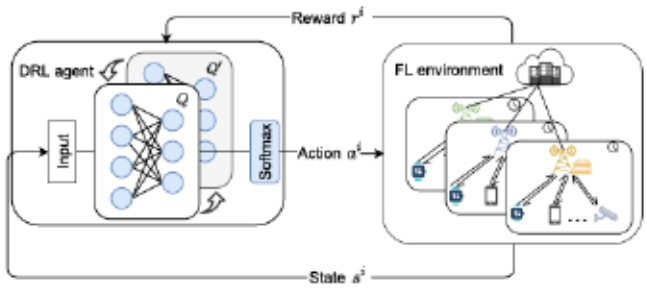


Fig. 6. The DRL agent interacting with the FL environment.

approximates to the optimal action-value function $Q^*(\mathrm{s}^i, a^i)$ [26]. DDQN introduces a double estimator $Q'(\mathrm{s}^i, a^i, \theta_i')$, which frozes every U updates, to stabilize the action-value function estimation.

To train the DRL agent, as depicted in Fig. 6, current state information s^i is fed into action-value function Q and then DQN generates action a^i as the staleness threshold for the edge node which is willing to join the FL model training. After interacting with the FL environment for several rounds, the DRL agent samples a few state-action pairs from the experience memory to solve (4) as

$$\arg\min_{\theta_i} \mathrm{L}(\theta_i) = (Y_i' - Q_\pi(\mathrm{s}^i, a^i, \theta_i))^2, \tag{27}$$

where the target Y_i' is defined as

$$Y_i' = r^i + \gamma Q'(\mathrm{s}^{i+1}, \arg\max_{a^{i+1}} Q(\mathrm{s}^{i+1}, a^{i+1}, \theta_i), \theta_i'), \tag{28}$$

where θ_i is the online parameters updated per time step and θ_i' is the parameters of double estimator Q'.

The action-value function $Q(\mathrm{s}^i, a^i)$ is updated by minimizing $\mathbb{L}(\theta_i)$ with gradient descent, i.e.,

$$\theta_{i+1} = \theta_i + \alpha_Q(Y_i' - Q(\mathrm{s}^i, a^i, \theta_i)\nabla_{\theta_i} Q(\mathrm{s}^i, a^i, \theta_i)), \tag{29}$$

where α_Q is a scalar step size. Besides, the classical ϵ-greedy policy is adopted in DDQN model training to aid exploration [26]. The details of the DRL-based staleness control process is elaborated in Algorithm 2.

The DRL agent is deployed in the scheduler component of the cloud server, thus the scheduler can inform an edge about the staleness threshold while distributing the latest global model to the edge who sends a check-in request.

C. Heterogeneity-Aware Association At Client-Edge Layer

Inspired by the convergence analysis in Section IV and the discussion in Section V-A, we identify the controllable factors for learning performance enhancement and aim to design a client-edge association mechanism that minimizes the edge-cloud model divergence. Due to the synchronous mechanism, the edge model aggregation can be conducted until the associated slowest client uploads its newly-updated model. Hence, the resource heterogeneity (e.g., response latency) among the clients should also be taken into consideration. We strike a nice balance between data heterogeneity of edges and resource heterogeneity

Algorithm 2: DDQN Training Procedure for Adaptive Staleness Control.

Input: discount factor γ, target accuracy Acc_{target}, experience memory maximum size B_{max}, update frequency of target network U.

```
Initialize experience memory E_exp
Initialize action-value network Q with initial weight θ
Initialize the target network Q' as a copy of θ
for each episode epi = 1, 2, 3, ... do
  for i = 1, 2, ..., I do
    Get current state s^i from the FL learning environment
    Generate action a^i based on ε-greedy policy and execute it
    Observe reward r^i and next state s̃^i
    Store the tuple (s^i, a^i, r^i, s̃^i) into E_exp
    if |E_exp| > B_max then
      Remove the oldest tuple
    end if
    Sample a minibatch of tuples (s, a, r, s̃) from E_exp
    Get target values with (28)
    Train and update the action-value network with the objective of (27) and (29)
    Update the target network as a copy of the weights of action-value network every U steps
    Get the accuracy Acc of the cloud model
    if Acc >= Acc_target then
      break out of current episode
    end if
  end for
end for
```

inherent in clients to establish a heterogeneity-aware client-edge association mechanism for fast and accurate cloud model learning.

For data heterogeneity in FL, we primarily investigate the label distribution skew which always exists in real-world applications [27]. As the edge-cloud model divergence is attributed to data heterogeneity between the edge node and the cloud server, we resort to Jensen-Shannon (JS) divergence [28], which is based on Kullback-Leibler (KL) divergence, to calculate the dissimilarity between two datasets. Considering two probability distributions P and P', the JS divergence between P and P' is defined as

$$JS(P||P') = \frac{1}{2}KL\left(P||\frac{P+P'}{2}\right) + \frac{1}{2}KL\left(P'||\frac{P+P'}{2}\right),$$
$$KL(P_1||P_2) = \sum_{x\in X} P_1(x) log\frac{P_1(x)}{P_2(x)}. \tag{30}$$

JS divergence has appealing properties of symmetry and normalized values between [0,1], which is contrast with unbounded KL divergence. $JS(P||P') = 0$ indicates identical distributions of P and P' while $JS(P||P') \to 1$ means the distributions are considered highly distant. When facing feature distribution skew, we can leverage FedBN [29] as the aggregation method to

help harmonizing local feature distributions in the collaborative training process, which would be an interesting future research direction.

For resource heterogeneity of clients, we first define the response latency for client k associated with edge node m as:

$$L^{m,k} = l^{k}_{comp} + l^{m,k}_{comm}, \tag{31}$$

where l^{k}_{comp} is the average value of the computation latency $C^{i,k}_{comp}$ for client k over the time span and $l^{m,k}_{comm}$ is the average value of the communication latency $C^{i,m,k}_{comm}$ between edge m and client k over the time span. Here, both l^{k}_{comp} and $l^{m,k}_{comm}$ can be obtained from the historical records. As a result, the latency for edge model aggregation can be formulated as

$$L^{m} = \max_{k \in \mathcal{C}^m} L^{m,k}, \tag{32}$$

indicating that one edge node should form its client cluster by selecting clients with lower response latency to mitigate straggler effect and accelerate model training process.

Before performing client-edge association, each of the edge nodes first probes the clients in its communication range to measure the response latency and collect the data distributions of the clients. The data distribution of a client records the proportion of samples for different classes. For instance, given an application with 4 distinct labels and a client dataset $\mathcal{D}_k$ that has one example with label 0, and two examples with label 2, the client's label distribution can be defined as $LD_k(\mathcal{D}_k) = [\frac{1}{3}, 0, \frac{2}{3}, 0]$. Note that label information of a client is only revealed in an aggregated format (see (33)) by using secure multiparty computation (e.g., privacy-preserving k-secure sum protocol [30]) and after noise is added, so no violation of individual label privacy happens.

After obtaining the response latency and label distribution of clients in the communication range, the edge node m can strike a balance between measured latency and edge-cloud model divergence by calculating the cost defined as

$$COST^{m}_{k} = L^{m,k} + \lambda JS \times \left(\frac{|\mathcal{D}^m| \cdot LD^m + |\mathcal{D}_k| \cdot LD_k}{|\mathcal{D}^m| + |\mathcal{D}_k|}, LD_{IID} \right), \tag{33}$$

where λ is a weighting parameter for the trade-off between resource heterogeneity and data heterogeneity. LD^m represents the label distribution on edge node m, the weighted average of the label distributions in current client cluster $\mathcal{C}^m$. LD_{IID} denotes the IID label distribution hold by the cloud server, Here, we consider the global data distribution is IID as the cloud server coordinates the model learning on multiple edge nodes, reaching a large amount of samples from different classes.

Hence, the edge node can conduct heterogeneity-aware client-edge association in the following two-way selection manner:

1) If the client cluster for edge node m is empty, the edge node selects the client with the lowest response latency from the unassociated clients in its communication range. Otherwise, the edge node will select client k with smallest cost $COST^{m}_{k}$
2) If one client is currently selected by multiple edge nodes, the client will choose an edge node randomly.

Algorithm 3: Heterogeneity-Aware Client-Edge Association.

Input: Label distribution of all the clients $\{LD_k\}_{k=1}^{N}$, the response latency $\{l^{m}_{k}\}_{k=1}^{N}$ for edge node $m \in \{1, 2, \ldots, M\}$.

Construct a client pool $ClientPool = \{1, 2, \ldots, N\}$
Construct empty client cluster for each edge node: $\mathcal{C}^m = \emptyset$ for $m \in \{1, 2, \ldots, M\}$, and empty edge set $S_k = \emptyset$ for each client $k \in \{1, 2, \ldots, N\}$
While $ClientPool \neq \emptyset$ **do**
  for $m \in \{1, 2, \ldots, M\}$ **do**
    if $\mathcal{C}^m = \emptyset$ **then**
      Choose client k with lowest response latency and set $\mathcal{C}^m = \{k\}$
      Remove client k from $ClientPool$
    else
      Choose client k by minimizing (33)
        $S_k = S_k \cup \{m\}$
    end if
  end for
  for $k \in \{1, 2, \ldots, N\}$ **do**
    if $S_k \neq \emptyset$ **then**
      Randomly select an edge node m from S_k
        $\mathcal{C}^m = \mathcal{C}^m \cup \{k\}$
      Remove client k from $ClientPool$ and clear S_k
    end if
  end for

Output: Client clusters $\mathcal{C}^m$ for $m \in \{1, 2, \ldots, M\}$.

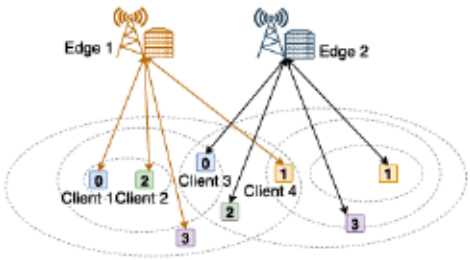


Fig. 7. The contour plot of response latency of the clients communicated to different edge nodes. The samples in each client are from only one class, represented by the number in the square. For edge 1, after selecting client 1 and client 2, it is better to choose client 4 rather than client 3, taking consideration of the trade-off between response latency and data distribution of the client.

The two-way selection procedure continues until all the clients are associated with one edge node. The detailed heterogeneity-aware client-edge association strategy is presented in Algorithm 3.

For a more intuitive illustration, as depicted in Fig. 7, after adding client 1 and client 2 into the client cluster of edge 1, it is better to choose client 4, rather than client 3, in order to make the data distribution of edge 1 close to the IID distribution. Similarly to many existing studies such as [31], [32], we consider that the clients are stationary or their locations change slowly during FL training process. This can be mainly motivated by

TABLE II
DATASETS AND THE CORRESPONDING MODELS

Dataset	Model	Parameter number
MNIST	LeNet	21,840
CIFAR10	ResNet-18	3,504,554
FEMNIST	CNN	214,590

that since FL requires intensive computing and frequent communication, most clients would like to participate FL when they are in stable conditions (e.g., when charging their batteries at home/office) [2]. For the case that clients' locations change fast, we can apply the adaptive client-edge association strategies to periodically update the clients' edge node selections. And some online optimization algorithms can be further leveraged to improve the performance of dynamic client-edge associations. Nevertheless, the theoretical analysis of such case is much more involved, and will be considered as a future work due to space limit.

In practice, with the label distributions of all the clients and the measured latency among clients and edges, we can easily get different client-edge association strategies under different λ. Hence, we can obtain the total JS divergence of all the edges under different λ and then select the client-edge association strategy with smaller JS divergence for fast FL model training. HiFlash can be seen as an enhanced HiFL approach equipped with adaptive staleness control and heterogeneity-aware client-edge association, hence we can choose either HiFL or HiFlash for efficient FL model training. In HiFlash, adaptive staleness control and heterogeneity-aware client-edge association are designed to alleviate the staleness issue in asynchronous aggregation at the cloud server and data heterogeneity among the edges, respectively. As a result, it is possible to combine each of these two parts with existing works with asynchronous FL for performance enhancement, by following the similar ideas developed in our paper.

VI. EXPERIMENTS

A. Simulation Settings

In order to gauge the effectiveness of our proposed algorithm, we conduct extensive evaluations in a simulated environment with 100 clients, 10 edge nodes and a cloud server. We consider image classification as the FL task and evaluate the performance of HiFL and HiFlash with three real-world datasets: MNIST [20], CIFAR10 [33] and FEMNIST [34]. As FEMNIST is a federated version of Extended MNIST dataset [35] whith 805,263 samples from 3,550 writers, we randomly select 100 writers as the clients to participate the model training of in our experiments. For the 10-class hand-written digit classification dataset MNIST, we use LeNet [36] as the model trained on the clients. For the CIFAR10 dataset, a standard ResNet-18 [37] model is adopted. For the 62-class hand-written digit classification dataset FEMNIST, we design a convolutional neural network (CNN) with 214,590 learning parameters as the learning model. The datasets and the corresponding models are summarized in Table II. All the experiments are conducted on one Tesla P100 12 GB GPU and the algorithms are implemented by Pytorch version 1.10.0.

For the local computation of the training on each client, we employ mini-batch Stochastic Gradient Descent (SGD) with a batch size of 60 for MNIST and FEMNIST, and 50 for CIFAR10, respectively. The initial learning rates are 0.01 for MNIST and FEMNIST, and 0.1 for CIFAR10 as in [13], both of which decay exponentially at a rate of 0.99 every 100 epochs. The number of local updates c for each client in one client-edge communication round is set to be 3 and the number of client-edge aggregations before pushing the edge model to the cloud server is set as $H \in \{1, 2, 3\}$.[3] The hyperparameters α and υ in coefficient α_τ can be determined by grid search in practice.

For DRL training, we set different threshold bounds ($\tau_{max} = 16$ for MNIST and FEMNIST datasets, and $\tau_{max} = 4$ for CIFAR10 dataset) for the three datasets as a bigger staleness threshold will result in much longer training time for the complicated CIFAR10 dataset. Hence, according to different action size, the DDQN model in the DRL agent, which is implemented by two two-layer multi-layer perceptron (MLP) networks, has 4,607 and 4,235 trainable parameters for MNIST (and FEMNIST), and CIFAR10 datasets, respectively. The output of the MLP network passing through a softmax layer becomes the probability of selecting a staleness threshold. The DDQN is lightweighted and each training iteration takes seconds on GPU.

Data Heterogeneity: For MNIST and CIFAR10 datasets, to simulate the data heterogeneity of clients in real world, we generate three kinds of data distributions for clients as below:

- *IID:* Each client is randomly assigned a uniform data distribution over 10 classes.
- *Non-IID(1):* Each client possesses only one random class of images.
- *Non-IID(2):* The samples in each client are assigned from two randomly selected classes.

While the FEMNIST dataset naturally falls in the following three data heterogeneity cases:

- *Label distribution skew:* The label distributions are totally different among the writers.
- *Feature distribution skew:* There is a natural feature distribution skew among different writers due to their different character features (e.g., stroke width, slant).
- *Quantity skew:* The samples in each client are ranging from [4,525].

The data distribution on an edge node can be obtained by calculating the weighted average of the data distributions of its associated clients. We can use JS divergence to measure the data heterogeneity of the edge nodes.

Resource Heterogeneity: The highly heterogeneous hardware resources (CPU, network connection) among clients can be reflected by the computing latency $C^{i,k}_{comp}$ and communication latency $C^{i,m,k}_{comm}$. For the computation ability of each client k, we assume $f_k \in [1, 2]$ GHz as the CPU cycle frequency and $\zeta_k = 20$

[3]The values of c and H depend on the computation budgets of the devices in practice. Due to the computing resource limitation of our research lab, we set small values for both c and H, but it is sufficient to evaluate the effectiveness of HiFlash.

cycles/bit as the number of CPU cycles to execute one bit. For the communication ability, the bandwidth $B_{k,m}$ is ranging from 1 MHz to 10 MHz when associated with different edge node m.

B. Metrics and Baselines

Performance Metrics. We consider *test accuracy*, *the number of communications with the cloud*, and *overall system cost* as three metrics for performance evaluation of HiFlash. Besides, we also calculate the *response latency* L^m and *waiting time* $L^m_{waiting}$ during synchronous client-edge aggregation to evaluate the effectiveness of client-edge association strategy. Here, the waiting time $L^m_{waiting}$ is measured by the average waiting time for the straggler in the clients associated to edge node m, denoted as

$$L^m_{waiting} = \frac{\sum_{k \in \mathcal{C}^m} (L^{m,k} - \min_{k \in \mathcal{C}^m} L^{m,k})}{|\mathcal{C}^m|}. \tag{34}$$

Baselines. We compare our proposed algorithm with both traditional centralized method and federated learning based schemes for performance evaluation:

- *Centralized Learning:* This scheme collects all the raw data to the cloud for training and provides an upper bound for model accuracy.
- *FedAvg [1]:* A cloud-based FL scheme with synchronous aggregation. Each time, it randomly selects 10 clients for local model training and global model aggregation.
- *FedAsync [8]:* A cloud-based asynchronous federated learning algorithm which updates the global model without waiting for straggling clients.
- *HierFAVG [13]:* A cloud-edge-client hierarchical FL scheme that performs synchronous update in both client-edge aggregation and edge-cloud aggregation. For fair comparison, 5 clients are randomly selected for edge model aggregation and 2 edge nodes contribute to the cloud model update in each global training round.
- *FedAT [15]:* A hierarchical FL scheme that combines synchronous intra-tier training and asynchronous cross-tier training. FedAT conducts client clustering based on their response latencies, without considering the data heterogeneity of the clients.

It is worthnoting that we adopt random staleness control mechanism with a fixed staleness threshold for the asynchronous FL schemes (e.g., FedAsync, FedAT and HiFL) for fair comparison with HiFlash. For MNIST dataset, we set a bigger value ($\tau_{max} = 16$) to facilitate more parallel model training and shorten the total training time. While for the complex CIFAR10 dataset, we set a smaller value ($\tau_{max} = 4$) to reduce resource waste as the local training of CIFAR10 dataset incurs high resource cost. When facing new datasets, we can determine the staleness threshold based on our preferences (e.g., cost efficiency, training time) of the FL learning task.

C. Experimental Results

Performance Evaluation for Various Hyperparameter Settings: As the global model updating of HiFL and HiFlash is controlled by α_τ which is related to the initial weight of edge model α and penalty coefficient υ, we first evaluate the test accuracy of HiFL under different settings of α and υ. As shown in Fig. 8, HiFL is robust and can converge within 1,000 communication rounds under the non-IID(2) data distribution with different mixing hyperparameter settings. However, a too large or too small value of α will result in a slow convergence speed. For example, when α is too large (e.g., $\alpha = 0.95$, $\alpha = 1.0$), the current global model fails to retain information about the global model from the previous round. While a smaller α (e.g., 0.5) will prevent the global model learning from the newly uploaded edge model.

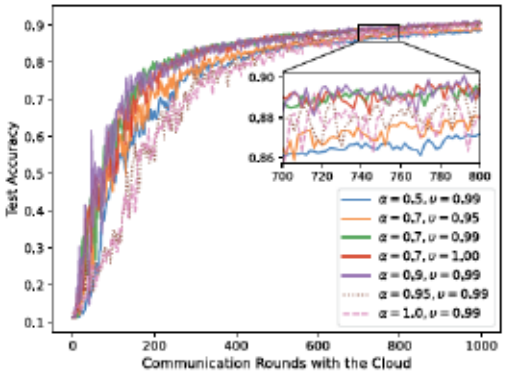
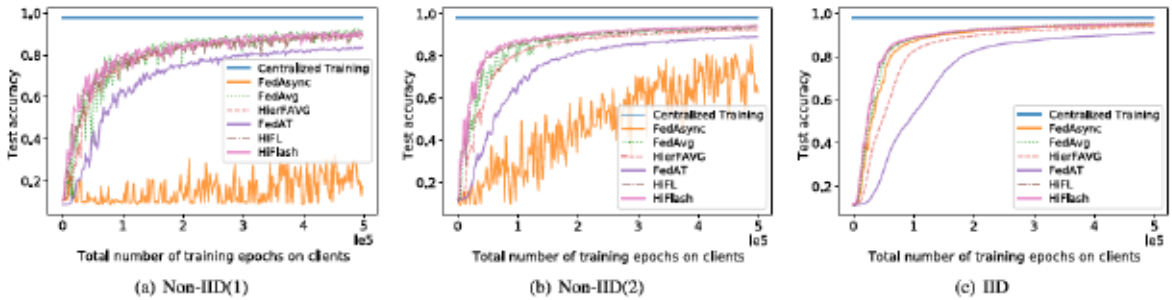


Fig. 8. Test accuracy of MNIST dataset under Non-IID(2) data distribution with different mixing hyperparameters.

Hence, we adopt grid search to find a proper choice for α and υ. We can see from Fig. 8 that the global model converges fast when $\alpha = 0.9$ and $\alpha = 0.7$, however, the test accuracy becomes more fluctuating with $\alpha = 0.9$ due to the deviation of the edge model. Besides, to deal with model staleness, a proper υ (e.g., 0.99) is effective for fast and stable model convergence. As the convergence results of different datasets under various mixing hyperparameter settings are similar, we set $\alpha = 0.7$ and $\upsilon = 0.99$ for the following experiments.

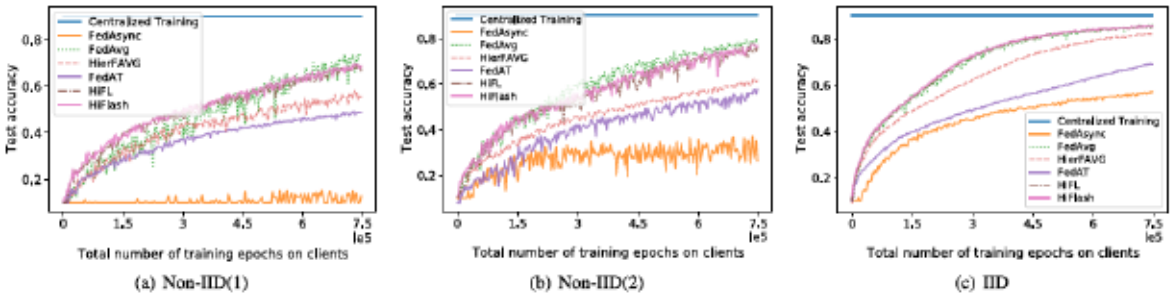
Model Accuracy and Computation Efficiency Evaluation: Considering the fact that hierarchical FL executes more local computations in one global round to reduce the costly communication with the cloud, we propose to evaluate the test accuracy with respect to the total number of training epochs on clients. As depicted in Figs. 9 and 10, we investigate the model accuracy and computation efficiency of different training methods under three kinds of data heterogeneity for MNIST and CIFAR10 datasets, respectively. As the centralized training method collects all the data to the cloud for model learning, it does not incur any computation cost on devices. Hence, we only use the test accuracy of centralized learning to provide an upper bound of model accuracy for other comparing methods.

We can see that by incorporating the merits of synchronous and asynchronous model aggregation and dampening the negative effect of model staleness, HiFL and HiFlash can achieve comparable training performance with FedAvg method in IID cases. For Non-IID cases, HiFL and HiFlash perform slightly less well than FedAvg when comparing test accuracy with respect to total number of training epochs on clients. This is



(a) Non-IID(1) (b) Non-IID(2) (c) IID

Fig. 9. Test accuracy of MNIST dataset under different data distributions w.r.t the total number of training epochs on the clients.



(a) Non-IID(1) (b) Non-IID(2) (c) IID

Fig. 10. Test accuracy of CIFAR10 dataset under different data distributions w.r.t the total number of training epochs on the clients.

because that the multiple rounds of client-edge aggregation in HiFL and HiFlash might lead to some degree of gradient divergence, and hence degrade the model performance. Moreover, as HiFL and HiFlash are designed with asynchronous model aggregation, they inevitably suffer from staleness effect, comparing with FedAvg.

HiFL and HiFlash perform better than other hierarchical FL methods (e.g., HierFAVG). Since hierarchical FL is designed to reduce the costly communication at the price of more local computations, HierFAVG, the extension of FedAvg in the hierarchical setting, is computationally inefficient comparing with FedAvg as shown in Figs. 9 and 10. This is because that HierFAVG performs fewer edge-cloud aggregations than FedAvg for the same amount of local training epochs. While with the asynchronous update mechanism which well balances the global model and the uploaded edge model in HiFL and HiFlash, we can see that the performance gap between HiFL and FedAvg narrows significantly, comparing with that between HierFAVG and FedAvg, indicating that the asynchronous aggregation in HiFL and HiFlash is more computationally efficient than other hierarchical FL schemes.

HiFL and HiFlash perform better than other synchronous or asynchronous based methods. For example, asynchronous FL methods (e.g., FedAsync and FedAT) have lower test accuracy than HiFL and HiFlash, since they ignore the negative impacts of biased data distribution and model staleness on the cloud model accuracy. The sharp oscillation in the curves of FedAsync algorithm attributes to the following two reasons: (1) the global model in FedAsync algorithm is updated once one client uploads its updated model without waiting for stragglers. This kind of asynchronous aggregation induces much uncertainty into the performance of the resulting global model, especially in Non-IID cases. While other algorithms fuse different client models to ensure the generalization ability of the global model; and (2) the staleness effect makes the convergence of FedAsync slower and causes the performance instability when facing large staleness.

Communication Efficiency Evaluation: We define the number of communications in FL process as the total communication number between edge nodes (or clients in two-layer FL frameworks) and the cloud server for model exchange. A smaller number of communications with the cloud indicates a smaller data size of models transferred to the cloud. To evaluate communication efficiency of HiFlash, we investigate the number of communications between the edges and the cloud to reach a target accuracy for all the FL based approaches.

As shown in Fig. 11, for MNIST dataset, the required communication numbers for different methods grow with the increase of target accuracy and data heterogeneity of clients. Except FedAsync algorithm, our proposed HiFL scheme is the most communication-efficient than other FL based methods regardless of data distribution and target accuracy. For example, HiFL can reduce the communication numbers by up to 31.9% than FedAT, 28.2% than HierFAVG and 77.6% than FedAvg on MNIST dataset with Non-IID(2) distribution and target accuracy of 90%. Although FedAsync can reach a target accuracy with fewer communication numbers than our proposed HiFL method in IID setting for MNIST dataset, it fails to deal with the data heterogeneity (e.g., Non-IID(2) and Non-IID(1) cases) inherent in the participating clients. For example, FedAsync can not achieve the target accuracy of 90% under Non-IID(2) distribution and even fails to reach the target accuracy of 70% in Non-IID(1)

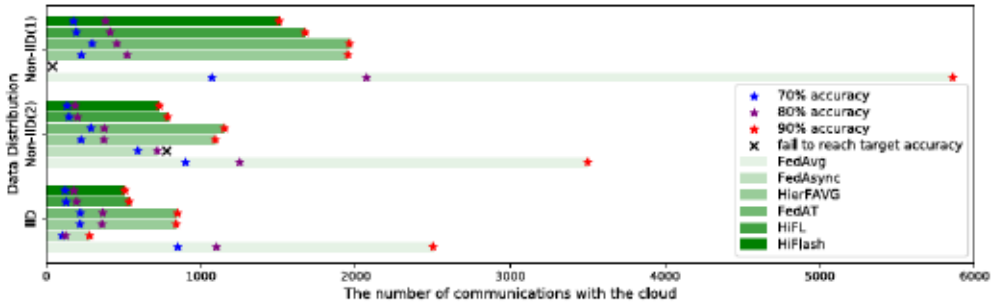


Fig. 11. The number of needed communications between the edge nodes and cloud server to reach a target accuracy for HiFlash comparing with different FL methods under different levels of data heterogeneity of MNIST dataset.

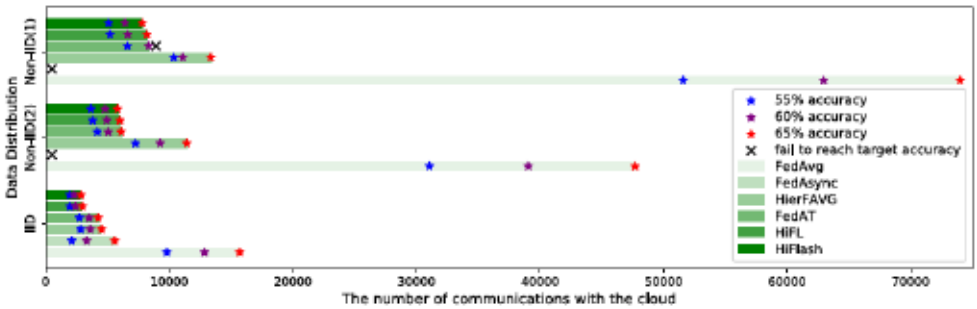


Fig. 12. The number of needed communications between the edge nodes and cloud server to reach a target accuracy for HiFlash comparing with different FL methods under different levels of data heterogeneity of CIFAR10 dataset.

case, indicating that FedAsync is not applicable in realistic FL scenarios where data is distributed in a Non-IID fashion.

The hierarchical FL methods (e.g., HiFL, HierFAVG) significantly reduce the costly communications with the cloud due to the client-edge aggregations. Moreover, the enhanced HiFlash framework, equipped with adaptive staleness control, can further accelerate the model training process and reduce the communication rounds with the cloud comparing with HiFL (e.g., 10% communication round reduction under Non-IID(1) data distribution case). This result is consistant with the convergence analysis that the model will converge within fewer communication rounds T_c by controlling τ in a smaller value. Thus, the client-edge aggregation and staleness control contribute to the communication efficiency of HiFL and HiFlash.

As for the comparison of FedAvg and HierFAVG, there are 10 clients communicating with the cloud in each training round for FedAvg, while 5 edges communicate with the cloud in HierFAVG method. Moreover, HierFAVG uses more local computation on the clients in each round to decrease the number of global training rounds, thus, HierFAVG is much better than FedAvg in terms of communication cost (e.g., the number of communications with the cloud).

For a more complicated dataset (i.e., CIFAR10), HiFlash can reach different target accuracies with the smallest communication rounds in all data distribution situations, comparing with all the FL based methods. As depicted in Fig. 12, HiFlash requires 6403 communication numbers with the cloud to reach the target accuracy of 60% in Non-IID(1) data distribution scenario, which is 23.07%, 42.23% and 89.82% smaller than FedAT, HierFAVG, and FedAvg, respectively. It shows that HiFlash outperforms current existing hierarchical FL algorithms, no matter asynchronous based FedAT or synchronous based HierFAVG method.

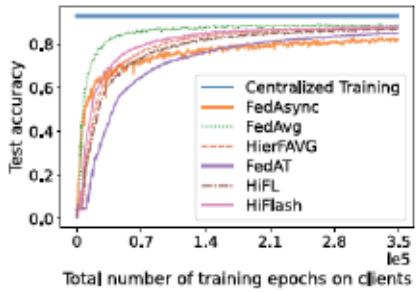


Fig. 13. Test accuracy of FEMNIST dataset w.r.t. the total number of training epochs on the clients.

Evaluation Results on FEMNIST Dataset: We also evaluate the performance of HiFlash under FEMNIST dataset where the data distributions of clients are naturally non-IID (in feature distribution, label distribution and quantity distribution). As shown in Fig. 13, HiFlash still performs better than other hierarchical FL methods (e.g., HierFAVG), which is similar with the experimental results under MNIST dataset. Moreover, the number of communications with the cloud of HiFlash approach is the smallest comparing with other FL methods, which can be seen in Fig. 14. The evaluation results show that our HiFlash approach can be applied in real-world datasets with skews in both label distribution and feature distribution.

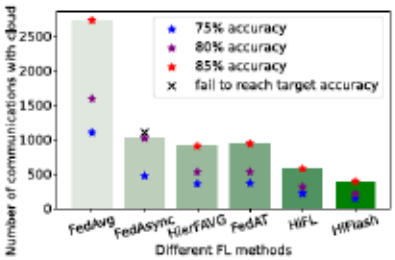


Fig. 14. Number of needed communications between edge nodes and cloud server to reach a target accuracy under FEMNIST.

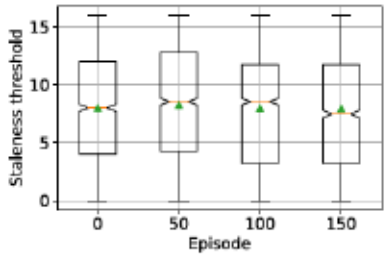


Fig. 16. The policy improvement process for adaptive staleness control with the increasing of DRL training episodes.

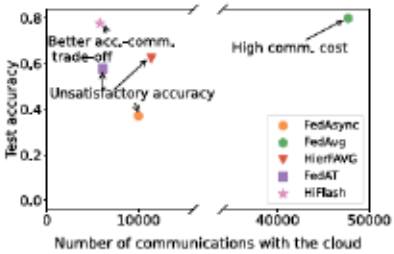


Fig. 15. HiFlash strikes a nice accuracy-communication trade-off comparing with other FL based approaches.

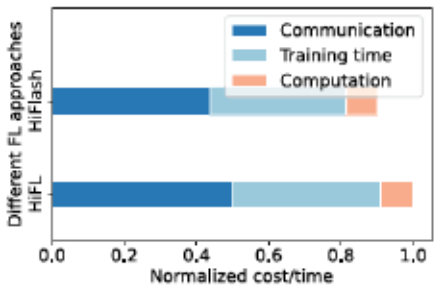


Fig. 17. The total system costs for HiFL and HiFlash.

Accuracy-Communication Trade-Off: To clearly show the superiority of HiFlash, we further investigate the accuracy-communication trade-off for different FL methods. Giving CIFAR10 dataset with Non-IID(2) distributions as an example, we plot the highest model accuracy and the communication rounds with the cloud in Fig. 15. We can see that the popular FedAvg approach suffers from high communication cost, while asynchronous based FedAsync approach and hierarchical FL methods (e.g., HierFAVG and FedAT) fails to achieve a satisfactory accuracy. In contrary, HiFlash is able to strike a nice balance between model accuracy and communication efficiency. Specifically, HiFlash significantly reduces the communication cost (e.g., 87% than FedAvg) with a slight model accuracy degradation (e.g., 2.1%). When comparing with hierarchical FL approaches (e.g., FedAT and HierFAVG), HiFlash is able to achieve more than 5% communication cost reduction and 16% model accuracy improvement. Furthermore, the superiority of HiFlash can be amplified as data heterogeneity increases (see Figs. 11 and 12).

The Effect of Staleness Threshold Control: The fast model convergence speed and high communication efficiency achieved by HiFlash are attributed to the well design of DRL-based adaptive staleness control that makes a wise decision according to the past experiences and current environment. As illustrated in Fig. 16, the DRL agent for adaptive staleness control improves the staleness threshold decision policy unremittingly as it interacts with the FL environment and learns from the DRL training episodes. We adopt boxplot to graphically depict the five-number summary of the distribution of the staleness threshold decisions in different training episodes, which consists of the smallest observation, lower quartile, median, upper quartile and largest observation. The upper quartile, median and lower quartile make up a box with compartments. The spacings between different parts indicate the variance and skew in the data distribution and the mean of staleness thresholds in shown with the green triangles. The decision policy in episode 0 is the random staleness control policy adopted in HiFL while the improved policy in episode 150 is utilized by the HiFlash approach. We can learn from the skewed data distribution that more decisions made in HiFlash choose a smaller staleness threshold compared with random decision policy, resulting in communication-efficient model training.

To compare the system costs (including computation cost of the clients, communication cost of the edges and training time of the cloud) of HiFL and HiFlash, we normalize the system cost to [0,1] by simply dividing the biggest value. With this normalization method, the computation cost and communication cost are scaled down accordingly. As shown in Fig. 17, the normalized cost of HiFL is 1, indicating that HiFL brings high system cost. While the lower system cost of HiFlash is credited to the effectiveness of our adaptive staleness control strategy design. It is worthnoting that although the DQN training in HiFlash brings additional computation overhead, it can be conducted on the cloud server with sufficient computation resources in an offline manner.

We also examine the policy differences with various reward designs by assigning different weights to computation cost, communication cost and training time (adjusting the values of σ_1 and σ_2). As shown in Fig. 18, when assigning higher weights to resource cost (e.g., computation and communication cost), the DRL agent for staleness control tends to choose smaller threshold in fear of severe staleness effect. While for a higher weight of training time, more decisions with bigger staleness threshold are made to facilitate model parallelism.

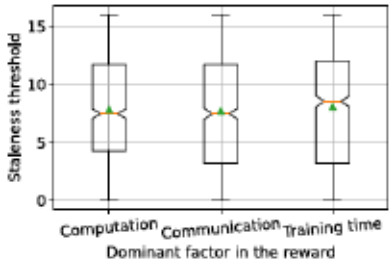


Fig. 18. Policy differences when facing various reward designs.

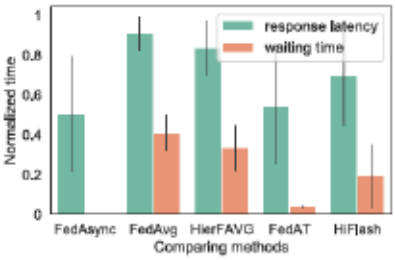


Fig. 19. Average response latency and waiting time for different FL methods.

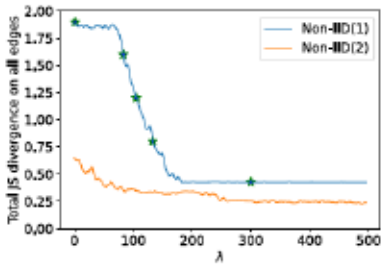


Fig. 20. JS divergence with varying λ under different data heterogeneity.

The Effect of Heterogeneity-Aware Client-Edge Association: Besides, we also evaluate the response latency $L^{m,k}$ and waiting time $L^{m}_{waiting}$ of the edge-associated clients in the client-edge aggregation phase. For HiFlash, we set $\lambda = 300$ (will be discussed in next paragraph) to form a more label-balanced dataset for each edge node. Thus, the average response latency may be longer due to the trade-off between data heterogeneity and latency reduction. As shown in Fig. 19, the average response latency and waiting time of FedAsync are the lowest since there is no need to wait for stragglers. FedAT, an asynchronous hierarchical FL scheme similar with our proposed HiFL, also has lower response latency and waiting time. However, it only focuses on latency reduction in client-edge association procedure without the consideration of data heterogeneity mitigation. Thus, both FedAsync and FedAT result in degraded accuracy as in Figs. 9 and 10 and more communications as in Figs. 11 and 12. While our proposed HiFlash can enforce a trade-off between the response latency and data heterogeneity, achieving a satisfactory model performance.

We further investigate the total JS divergence on all the edges nodes, the resulted average response latency, and the model accuracy of HiFlash with varying λ. As depicted in Fig. 20,

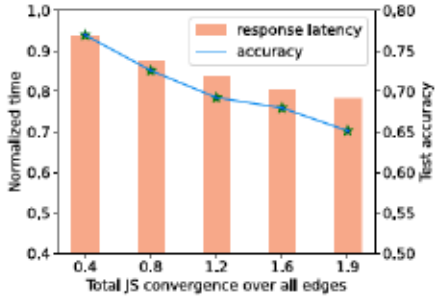


Fig. 21. Policy differences when facing various reward designs.

a bigger λ denotes that we are more concerned about the data distributions on the edge nodes compared with response latency, thus the total JS divergence decreases with the increasing λ in both Non-IID(1) and Non-IID(2) cases. In Fig. 21, a biased edge data distribution (larger JS divergence) will cause significant accuracy degradation. For example, the achievable accuracy drops from 77% to 65% when the total JS divergence over all edge nodes increases from 0.4 to 1.9. By controlling the parameter λ, our proposed HiFlash can be flexible for edge nodes to strike a balance between response latency and model accuracy.

VII. Related Work

A. Communication-Efficient Federated Learning

Classical two-layer FL frameworks (e.g., FedAvg [1] and FedAsync [8]) inevitably suffer from excessive communication overhead and network congestion in large-scale distributed machine learning, due to massive model exchanges between clients and central server. To reduce the bits on gradient exchanges in FL, techniques such as neural network pruning [38], weight quantization [39], message sparsification [40] and knowledge distillation [41] focus on ML model compression to reduce the amount of transmitted information while maintaining the high learning performance.

To further improve communication efficiency, hierarchical FL is proposed by introducing an edge layer, which leverages edge nodes as intermediaries to perform partial model aggregation with efficient client-edge communication, and thus relieves core network transmission overhead in the cloud server [14]. For example, Liu et al. propose HierFAVG that performs two level of synchronous model aggregation by extending the conventional FedAvg algorithm to the hierarchical setting [13]. However, a severe straggler problem would incur in HierFAVG due to the nature of synchronous model aggregation. Chai et al. present FedAT, a novel FL method that synergistically combines synchronous intra-tier training and asynchronous cross-tier training to improve the convergence speed and reduce communication cost [15]. Nevertheless, FedAT uses a weighted sum of all the latest edge models for global model update, which is different from our asynchronous aggregation mechanism. Moreover, it ignores the staleness effect, which is inevitable in asynchronous aggregation.

B. Model Staleness Control

Staleness effect is a common challenge in the asynchronous model aggregation. Most of existing FL solutions tolerate staleness by dampening the impacts of stale result that is computed on an outdated global model version. For example, Zhang et al. propose a staleness-aware async-SGD algorithm in which the learning rate is modulated according to the gradient staleness [42]. Xie et al. use a weighted average for global model update and introduce a mixed hyperparameter to adaptively control the error caused by staleness [8]. An gradient correction term is designed to compensate the staleness in [43]. Nevertheless, these approaches only focus on the negative impact of staleness on model accuracy and convergence speed. The system efficiency, such as computation/communication cost, and training time, is less considered.

A stale model may marginally contribute to the global model, but cause large resource waste and time consumption without timely terminating model training and uploading. To address this issue, FedSA [44], a staleness-aware asynchronous FL algorithm sets a staleness threshold for each participating client based on its computing speed. However, this approach ignores the communication cost and the training time in the whole process. Besides, the staleness threshold in FedSA is fixed for each client, which can not well adapt to the realistic dynamic environment. Zhang et al. propose a clustered semi-asynchronous federated learning (CSAFL) approach [45], which alleviates the model staleness problem by dividing clients with different learning objectives into multiple groups and limiting the model delay. However, CSAFL aims to learn personalized models, i.e., different group models for different client groups, while the HiFlash approach has a different goal and aims to learn a common global model that merits the commonality of the global knowledge sharing based on all the local data generated on clients.

Staleness control in asynchronous FL is similar with the client selection in synchronous FL, as they are both decision making problems. Hence, the multi-arm bandit based selection methods, deep reinforcement learning algorithms used in client selection problem can also be adopted in staleness control. However, client selection approaches in synchronous FL usually strike a balance among model accuracy, system efficiency and selection fairness. As staleness control problem does not need to ensure fairness for each threshold choice due to asynchronous aggregation which only involves one client, the client selection approaches (e.g., [46], [47]) with fairness consideration can not be utilized in staleness control problem. Moreover, in client selection approaches, the server often needs to send the global model to all the clients for local loss estimation before making selection decisions (e.g., [47], [48]), which is not required for staleness control.

C. Client-Edge Association in Hierarchical FL

The client-edge association strategy in hierarchical FL has a significant impact on model learning performance due to the data and resource heterogeneity across the dispersed clients [49]. HierFAVG ignores the varying training speed of the distributed clients and randomly groups them into different clusters, which may prolong the communication time of each training round [13]. Considering the straggler effect, both TiFL [49] and FedAT [15] divide the clients into different tiers based on the measured latency so that the resource heterogeneity can be mitigated [49]. To tackle the data heterogeneity in FL, Duan et al. propose FedGroup, which clusters clients into multiple groups based on the cosine similarity of their parameter updates [50]. In FedCluster, the authors provide several representative clustering approaches, including random uniform clustering, timezone-based clustering and availability-based clustering, to support for various application scenarios [51]. Unfortunately, current client-edge association schemes only focus on one dimension of heterogeneity, without considering the trade-off between data and resource heterogeneities.

Existing convergence analysis of hierarchical FL schemes mainly focus on FL with two levels of synchronous model aggregations. For example, the convergence of HierFAVG measures two-level of non-IIDness (i.e., the client level and the edge level) for data distribution in the hierarchical system, and provides qualitative guidelines on picking the aggregation frequencies at two levels [13]. While the theoretical analysis in [21] further proves that reducing the non-IIDness at the edge level is important for the model convergence. Although FedAT [15] is designed with synchronous client-edge aggregation and asynchronous edge-cloud aggregation, the asynchronous aggregation mechanism in FedAT is a weighted sum of all the latest edge models, which is different from our asynchronous update mechanism in (8). Moreover, the convergence analysis of FedAT ignores the staleness effect and hence fails to provide insights for staleness effect alleviation. Nevertheless, our convergence analysis for HiFL considers both staleness introduced by asynchronous model aggregation and non-IIDness inherent in FL, and further draws some insights for staleness control and non-IIDness reduction to achieve a fast convergence rate and a low convergence bound.

VIII. Conclusion

In this paper, we resort to HiFL, a hierarchical FL approach that synergistically employs synchronous client-edge model aggregation and asynchronous edge-cloud model aggregation for communication-efficient model learning. Based on the convergence analysis of HiFL, we identify the controllable factors for model convergence and further advocate HiFlash, an enhanced HiFL with adaptive staleness control and heterogeneity-aware client-edge association, for large-scale deployment in reality. We propose a DRL-based staleness threshold decision algorithm for accurate and cost-efficient FL model learning. To tackle the inherent resource and data heterogeneity among clients, we design a heterogeneity-aware client-edge association strategy that strikes a nice balance between communication latency and the heterogeneity of edge data distributions. Our empirical evaluation based on three image classification datasets validates our theoretical analysis, and demonstrates that HiFlash achieves satisfactory prediction performance for different levels of data heterogeneity and is communication-efficient compared with existing FL methods.

REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Aguera y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[2] P. Kairouz et al., "Advances and open problems in federated learning," *Foundations Trends Mach. Learn.*, vol. 14, no. 1–2, pp. 1–210, 2021.

[3] A. Differential P. Team, "Learning with privacy at scale," *Apple Mach. Learn. J.*, vol. 1, no. 8, 2017. [Online]. Available: https://machinelearning.apple.com/2017/12/06/learning-with-privacy-at-scale.html

[4] T. Yang et al., "Applied federated learning: Improving Google keyboard query suggestions," 2018, *arXiv:1812.02903*.

[5] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 10, pp. 4229–4238, Oct. 2020.

[6] S. Wang et al., "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 63–71.

[7] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, "On the convergence of federated optimization in heterogeneous networks," 2018, *arXiv:1812.06127*.

[8] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," in *Proc. NeurIPS Workshop Optim. Mach. Learn.*, 2020, pp. 1–11.

[9] W. Y. Bryan Lim et al., "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tut.*, vol. 22, no. 3, pp. 2031–2063, Third Quarter 2020.

[10] W. Wu, L. He, W. Lin, and R. Mao, "Accelerating federated learning over reliability-agnostic clients in mobile edge computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1539–1551, Jul. 2021.

[11] M. Xu et al., "From cloud to edge: A first look at public edge platforms," in *Proc. ACM Internet Meas. Conf. Virtual Event*, 2021, pp. 37–53.

[12] J. Yuan, M. Xu, X. Ma, A. Zhou, X. Liu, and S. Wang, "Hierarchical federated learning through LAN-WAN orchestration," 2020, *arXiv:2010.11612*.

[13] L. Liu, J. Zhang, S. H. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *Proc. IEEE Int. Conf. Commun.*, 2020, pp. 1–6.

[14] S. Luo, X. Chen, Q. Wu, Z. Zhou, and S. Yu, "HFEL: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning," *IEEE Trans. Wirel. Commun.*, vol. 19, no. 10, pp. 6535–6548, Oct. 2020.

[15] Z. Chai, Y. Chen, L. Zhao, Y. Cheng, and H. Rangwala, "FedAT: A communication-efficient federated learning method with asynchronous tiers under non-IID data," 2020, *arXiv:2010.05958*.

[16] V. Mnih et al., "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[18] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[19] G. Damaskinos, R. Guerraoui, A.-M. Kermarrec, V. Nitu, R. Patra, and F. Taiani, "FLeet: Online federated learning via staleness awareness and performance prediction," in *Proc. 21st Int. Middleware Conf.*, 2020, pp. 163–177.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[21] J.-W. Lee, J. Oh, Y. Shin, J.-G. Lee, and S.-Y. Yoon, "Accurate and fast federated learning via IID and communication-aware grouping," 2020, *arXiv:2012.04857*.

[22] J. Feng, L. Liu, Q. Pei, and K. Li, "Min-max cost optimization for efficient hierarchical federated learning in wireless edge networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2687–2700, Nov. 2022.

[23] Y. Zhan, P. Li, and S. Guo, "Experience-driven computational resource allocation of federated learning by deep reinforcement learning," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2020, pp. 234–243.

[24] M. H. Nguyen, N. H. Tran, Y. Tun, Z. Han, and C. Hong, "Toward multiple federated learning services resource sharing in mobile edge networks," *IEEE Trans. Mobile Comput.*, vol. 22, no. 01, pp. 541–555, Jan. 2023.

[25] Y. Sun, J. Shao, Y. Mao, J. H. Wang, and J. Zhang, "Semi-decentralized federated edge learning for fast convergence on non-IID data," in *Proc. IEEE Wirel. Commun. Netw. Conf.*, 2022, pp. 1898–1903.

[26] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[27] J. Zhang et al., "Federated learning with label distribution skew via logits calibration," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 26311–26329.

[28] A. P. Majtey, P. W. Lamberti, and D. P. Prato, "Jensen-shannon divergence as a measure of distinguishability between mixed quantum states," *Phys. Rev. A*, vol. 72, no. 5, 2005, Art. no. 052310.

[29] X. Li, M. Jiang, X. Zhang, M. Kamp, and Q. Dou, "FedBN: Federated learning on non-IID features via local batch normalization," in *Proc. Int. Conf. Learn. Representations*, 2021, pp. 1–27.

[30] R. Sheikh, B. Kumar, and D. K. Mishra, "Privacy preserving k secure sum protocol," 2009, *arXiv:0912.0956*.

[31] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *Proc. IEEE Int. Conf. Commun.*, 2019, pp. 1–7.

[32] G. Damaskinos, R. Guerraoui, A.-M. Kermarrec, V. Nitu, R. Patra, and F. Taiani, "FLeet: Online federated learning via staleness awareness and performance prediction," *ACM Trans. Intell. Syst. Technol.*, vol. 13, no. 5, pp. 1–30, 2022.

[33] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," *Tech. Report, Univ. Toronto*, 2009.

[34] S. Caldas et al., "LEAF: A benchmark for federated settings," 2018, *arXiv:1812.01097*.

[35] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "EMNIST: Extending MNIST to handwritten letters," in *Proc. Int. Joint Conf. Neural Netw.*, 2017, pp. 2921–2926.

[36] Y. LeCun et al., "Handwritten digit recognition with a back-propagation network," in *Proc. Adv. Neural Inf. Process. Syst.*, 1990, pp. 396–404.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[38] Y. Jiang et al., "Model pruning enables efficient federated learning on edge devices," 2019, *arXiv:1909.12326*.

[39] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1709–1720.

[40] W. Luping, W. Wei, and L. Bo, "CMFL: Mitigating communication overhead for federated learning," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 954–964.

[41] J.-H. Ahn, O. Simeone, and J. Kang, "Wireless federated distillation for distributed edge learning with heterogeneous data," in *Proc. IEEE 30th Annu. Int. Symp. Pers. Indoor Mobile Radio Commun.*, 2019, pp. 1–6.

[42] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware async-SGD for distributed deep learning," in *Proc. 25th Int. Joint Conf. Artif. Intell.*, 2016, pp. 2350–2356.

[43] L. Zhu et al., "Delayed gradient averaging: Tolerate the communication latency for federated learning," in *Proc. 35th Conf. Neural Inf. Process. Syst.*, 2021, pp. 29995–30007.

[44] M. Chen, B. Mao, and T. Ma, "FedSA: A staleness-aware asynchronous federated learning algorithm with non-IID data," *Future Gener. Comput. Syst.*, vol. 120: pp. 1–12, 2021.

[45] Y. Zhang et al., "CSAFL: A clustered semi-asynchronous federated learning framework," in *Proc. Int. Joint Conf. Neural Netw.*, 2021, pp. 1–10.

[46] T. Huang, W. Lin, L. Shen, K. Li, and A. Y. Zomaya, "Stochastic client selection for federated learning with volatile clients," *IEEE Internet Things J.*, vol. 9, no. 20, pp. 20055–20070, Oct. 2022.

[47] Y. J. Cho, J. Wang, and G. Joshi, "Towards understanding biased client selection in federated learning," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2022, pp. 10351–10375.

[48] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Oort: Efficient federated learning via guided participant selection," in *Proc. 15th USENIX Symp. Operating Syst. Des. Implementation*, 2021, pp. 19–35.

[49] Z. Chai et al., "TiFL: A tier-based federated learning system," in *Proc. 29th Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2020, pp. 125–136.

[50] M. Duan et al., "FedGroup: Efficient federated learning via decomposed similarity-based clustering," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl. Big Data Cloud Comput. Sustain. Comput. Commun. Social Comput. Netw.*, 2021, pp. 228–237.

[51] C. Chen, Z. Chen, Y. Zhou, and B. Kailkhura, "FedCluster: Boosting the convergence of federated learning via cluster-cycling," 2020, *arXiv:2009.10748*.

Qiong Wu received the BS and ME degrees from the School of Data and Computer Science, Sun Yat-sen University (SYSU), Guangzhou, China, in 2017 and 2019, respectively. She is currently working toward the PhD degree with the School of Computer Science and Engineering, Sun Yat-sen University. Her primary research interests include social data analysis, mobile edge computing, and federated learning.

Xu Chen (Senior Member, IEEE) received the PhD degree in information engineering from the Chinese University of Hong Kong, in 2012. He is a full professor with Sun Yat-sen University, Guangzhou, China, and the vice director of National and Local Joint Engineering Laboratory of Digital Home Interactive Applications. He worked as a postdoctoral research associate with Arizona State University, Tempe, from 2012 to 2014, and a Humboldt Scholar fellow with the Institute of Computer Science, University of Goettingen, Germany, from 2014 to 2016. He received the prestigious Humboldt research fellowship awarded by the Alexander von Humboldt Foundation of Germany, 2014 Hong Kong Young Scientist Runner-up Award, 2017 IEEE Communication Society Asia-Pacific Outstanding Young Researcher Award, 2017 IEEE ComSoc Young Professional Best Paper Award, Honorable Mention Award of 2010 IEEE international conference on Intelligence and Security Informatics (ISI), Best Paper Runner-up Award of 2014 IEEE International Conference on Computer Communications (INFOCOM), and Best Paper Award of 2017 IEEE Intranational Conference on Communications (ICC). He is currently an area editor of the *IEEE Open Journal of the Communications Society*, an associate editor of the *IEEE Transactions Wireless Communications*, *IEEE Internet of Things Journal* and *IEEE Journal on Selected Areas in Communications* (JSAC) Series on Network Softwarization and Enablers.

Tao Ouyang received the BS degree from the School of Information Science and Technology, University of International Relations, Beijing, China, in 2017, and the ME degree from the School of Computer Science and Engineering, Sun Yat-sen University (SYSU), Guangzhou, China, in 2019. He is currently working toward the the PhD degree with the School of Computer Science and Engineering, SYSU. His research interests include mobile edge computing, online learning, and optimization.

Zhi Zhou (Member, IEEE) received the BS, ME, and PhD degrees from the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, in 2012, 2014, and 2017, respectively. He is currently an associate professor with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. Since 2016, he has been a visiting scholar with the University of Göttingen, Göttingen, Germany. His research interests include edge computing, cloud computing, and distributed systems. He was nominated for the 2019 CCF Outstanding Doctoral Dissertation Award, the sole recipient of 2018 ACM Wuhan & Hubei Computer Society Doctoral Dissertation Award, and the recipient of the Best Paper Award of IEEE UIC 2018.

Xiaoxi Zhang (Member, IEEE) received the BE degree in electronics and information engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2013, and the PhD degree in computer science from the University of Hong Kong, Hong Kong, in 2017. She is currently an associate professor with the School of Computer Science and Engineering, Sun Yat-sen University (SYSU), Guangzhou, China. Before joining SYSU, she was a postdoctoral researcher with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA. Her research interests include optimization and algorithm design for networked systems, including cloud and edge computing networks, NFV systems, and distributed machine learning systems.

Shusen Yang (Senior Member, IEEE) received the PhD degree in computing from Imperial College London, in 2014. He is a professor and director of the National Engineering Laboratory for Big Data Analytics, and deputy director of Ministry of Education(MoE) Key Lab for Intelligent Networks and Network Security, Xi'an Jiaotong University (XJTU), Xi'an, China. Before joining XJTU, he worked as a lecturer (assistant professor) with the University of Liverpool from 2015 to 2016, and a research associate with Intel Collaborative Research Institute (ICRI) on sustainable connected cities from 2013 to 2014. He is a DAMO Academy Young Fellow, and an honorary research fellow with Imperial College London. He is a member of ACM. His research interests include focuses on distributed systems and data sciences, and their applications in industrial scenarios, including data-driven network algorithms, distributed machine learning, Edge-Cloud intelligence, industrial internet, and industrial intelligence.

Junshan Zhang (Fellow, IEEE) received the PhD degree from the School of ECE, Purdue University, in Aug. 2000. He is a professor with the ECE Department, University of California Davis. He was on the faculty of the School of ECEE. Arizona State University from 2000 to 2021. His research interests include fall in the general field of information networks and data science, including edge intelligence, reinforcement learning, network optimization and control, game theory, with applications in connected and automated vehicles, 5G and beyond, wireless networks, IoT data privacy/security, and smart grid. He is a recipient of the ONR Young Investigator Award in 2005 and the NSF CAREER award in 2003. He received the IEEE Wireless Communication Technical Committee Recognition Award in 2016. His papers have won a few awards, including the Best Student paper at WiOPT 2018, the Kenneth C. Sevcik Outstanding Student Paper Award of ACM SIGMETRICS/IFIP Performance 2016, the Best Paper Runner-up Award of IEEE INFOCOM 2009 and IEEE INFOCOM 2014, and the Best Paper Award at IEEE ICC 2008 and ICC 2017. Building on his research findings, he co-founded Smartiply Inc in 2015, a edge Computing startup company delivering boosted network connectivity and embedded artificial intelligence for IoT applications. He is currently serving as the editor-in-chief for *IEEE Transactions on Wireless Communication* and an editor-at-large for *IEEE/ACM Transactions on Networking*. He was TPC co-chair for a number of major conferences in communication networks, including IEEE INFOCOM 2012 and ACM MOBIHOC 2015. He was the general chair for ACM/IEEE SEC 2017, WiOPT 2016, and IEEE Communication Theory Workshop 2007. He was a Distinguished Lecturer of the IEEE Communications Society. He was an editor for the Computer Network journal, and an editor of IEEE Wireless Communication Magazine.